PENERAPAN ALGORITME *PARTICLE SWARM OPTIMIZATION – LEARNING VECTOR QUANTIZATION* (PSO-LVQ) PADA KLASIFIKASI DATA IRIS

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Ilham Romadhona
NIM: 145150200111192

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

PENERAPAN ALGORITME *PARTICLE SWARM OPTIMIZATION – LEARNING VECTOR QUANTIZATION* (PSO-LVQ) PADA KLASIFIKASI DATA IRIS

SKRIPSI

untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Ilham Romadhona
NIM: 145150200111192

Skripsi ini telah diuji dan dinyatakan lulus pada
26 Juli 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Imam Cholissodin, S.Si., M.Kom.	Drs. Marji, M.T.
NIK. 201201 850719 1 001	NIP . 19670801 199203 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 26 Juli 2018

Ilham Romadhona

145150200111192

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Allah SWT, karena atas anugerah serta limpahan rahmat-Nya sehingga, sehingga penulis dapat menyelesaikan skripsi dengan judul **"***Penerapan Algoritme Particle Swarm Optimization-Learning Vector Quantization (PSO-LVQ) Pada Klasifikasi Data Iris***"** ini. Skripsi ini disusun sebagai syarat memperoleh gelar sarjana pada Program Studi Teknik Informatika, Fakultas Ilmu Komputer Universitas Brawijaya.

Dalam proses penyelesaian skripsi ini penulis mendapatkan banyak bantuan, baik bantuan moral maupun materiil dari berbagai pihak. Oleh karena itu, penulis mengucapkan banyak terimakasih kepada:

1. Imam Cholissodin, S.Si., M.Kom. selaku Pembimbing pertama yang telah memberikan bimbingan, arahan, ilmu, dan masukan dalam penyelesaian skripsi ini.
2. Drs. Marji, M.T. selaku Pembimbing kedua yang juga telah memberikan bimbingan, arahan, ilmu, dan masukan dalam penyelesaian skripsi ini.
3. Bapak dan Ibu dosen yang telah mendidik dan memberikan ilmu selama Penulis menempuh pendidikan di Fakultas Ilmu Komputer Universitas Brawijaya.
4. Kedua Orang Tua, adik dan keluarga besar yang telah mendukung penulis dengan segala usahanya, mulai dari doa, materi, dukungan moral, semangat hidup, dan tauladan yang semata-mata untuk keberhasilan penulis.
5. Teman-teman Program Studi Teknik Informatika yang selalu memberikan semangat, dukungan, dan kebersamaan selama Penulis menempuh pendidikan di Fakultas Ilmu Komputer Universitas Brawijaya.
6. Teman-teman terdekat yaitu Anas Fitrah Abadi, Gholib Tamam Fauzi, Silviyah Mujiono, Nikmatul Maula dan teman-teman yang tidak dapat disebutkan satu per satu yang selalu memberikan bantuan, motivasi, semangat dan dukungan selama Penulis menyelesaikan skripsi ini.
7. Teman-teman dari *Basic Computing Community* (BCC) yang telah memberikan semangat, ilmu serta mendukung Penulis selama menempuh pendidikan di Fakultas Ilmu Komputer Universitas Brawijaya.
8. Teman-teman dari grup *Gazebo Army* (GA) yang selau menghibur, memberi semangat dan selalu mendukung selama Penulis menempuh pendidikan di Fakultas Ilmu Komputer Universitas Brawijaya.
9. Semua pihak yang tidak dapat disebutkan satu per satu, yang telah membantu dan terlibat baik secara langsung maupun tidak langsung dalam penulisan skripsi ini.

Penulis menyadari bahwa skripsi ini tidak lepas dari kesalahan dan kekurangan. Oleh karena itu, Penulis bersedia menerima kritik dan juga saran yang membangun untuk memperbaiki diri. Penulis berharap semoga skripsi ini dapat memberi manfaat.

Malang, 26 Juli 2018

Penulis

romadona.ilham2@gmail.com

ABSTRAK

Bunga Iris saat ini telah mudah dijumpai diberbagai penjuru dunia dengan spesies yang bermacam-macam. Dalam bahasa Yunani Iris merupakan sosok dari dewi pelangi karena spesies Iris sendiri telah mencapai 260 hingga 300 macam spesies dengan warna bunga yang berwarna-warni dan mencolok. Karena banyaknya spesies Iris tersebut, maka dibutuhkan klasifikasi dalam membedakan spesies pada bunga Iris. Untuk menyelesaikan permasalahan tersebut, digunakan *algoritme Learning Vector Quantization* (LVQ) yang nantinya akan dioptimasi menggunakan algoritme *Particle Swarm Optimization* (PSO) dengan melakukan klasifikasi pada spesies Iris Sentosa, Iris Virginica dan VersiColor yang mana spesies tersebut telah didata sebelumnya pada dataset Iris. Hasil dari penelitian ini selanjutnya dibandingkan dengan klasifikasi menggunakan algoritme LVQ. Didapatkan rata-rata akurasi menggunakan algoritme PSO-LVQ sebesar 93,334%, sedangkan rata-rata akurasi menggunakan algoritme LVQ sebesar 84,268%. Selisih rata-rata akurasi yang didapatkan sebesar 9,066% yang menandakan algoritme PSO-LVQ memberikan peningkatan hasil yang cukup baik dibandingkan algoritme LVQ.

Kata kunci: *Dataset* Iris, Optimasi, *Particle Swarm Optimization, Learning Vector Quantization*

ABSTRACT

Currently Iris flowers are easily found in around the world with various species. In Greek Iris mean the goddess of the rainbow because Iris species has reached 260 to 300 various species with colorful and light flowers. Because of the large number of Iris species, it is necessary to classify the Iris species. To solve the problem, used the Learning Vector Quantization (LVQ) algorithm which will be optimization using the Particle Swarm Optimization (PSO) algorithm was used to classify species into Sentosa Iris, Virginica Iris and Versicolor Iris category where the species previously recorded on Iris dataset. Then the result of this study was compared with the classification using LVQ algorithm. The average accuracy obtained with PSO-LVQ algorithm is 93.334%, whereas the average accuracy with LVQ algorithm is 84.268%. The differece in accuracy is 9.066% it is mean PSO-LVQ algorithm give more a good provides result than LVQ algorithm.

Keywords: *Iris Dataset, Optimization, Particle Swarm Optimization, Learning Vector Quantization*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR PERSAMAAN

DAFTAR KODE PROGRAM

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Iris merupakan spesies tumbuhan di zaman ini telah banyak kita jumpai di berbagai penjuru dunia. Pada awalnya Iris hanya tumbuh di Eropa, Asia Barat, Afrika di bagian utara, dan daerah medeterania. Iris berasal dari salah satu bahasa Yunani untuk dewi pelangi, hal tersebut dapat dibuktikan dengan banyaknya spesies yang mencapai 260 hingga 300 macam spesies dengan warna bunga yang mencolok (Kamenetsky dan Hiroshi, 2013). Dari banyaknya spesies tersebut terdapat beberapa spesies yang memiliki kemiripan seperti Iris Sentosa, Iris VersiColor, dan Iris Virginica. Dari beberapa spesies yang memiliki kemiripan tersebut perlu dilakukan klasifikasi untuk membedakannya.

Klasifikasi merupakan proses dalam mencari sebuah informasi dari kumpulan data yang bertujuan memprediksi label atau kelas dari objek dalam suatu objek (Han dan Kamber, 2006). Metode-metode pengklasifikasian banyak sekali, salah satunya *Neural Network. Neural Network* (*NN*) atau umumnya Jaringan Syaraf Tiruan (JST) merupakan model komputasi yang memiliki struktur dan fungsional seperti jaringan syaraf manusia. *Neural Network* memiliki proses distribusi secara paralel yang berkecenderungan untuk menyimpan sebuah pengetahuan dari proses pembelajaran (Haykin, 1994). Secara umum *Neural Network* terdiri dari beberapa *input layer, beberapa hidden layer,* dan *output layer. Output layer* merupakan hasil permrosesan dari *Neural Network* yang didapat dari proses pembelajaran dari bobot awal yang telah diberikan pada *input layer*.

Banyak sekali metode-metode *Neural Network* yang dapat kita gunakan seperti *Hebbian Learning, Backpropagation*, dan *Learning Vector Quatization*. Salah satu algoritme *Neural Network adalah Learning Vector Quantization* (LVQ). *Learning Vector Quantization* merupakan metode pengklasifikasian pola yang mana *output* setiap unit mewakili kategori tertentu (Kohonen, 1990). *Learnig Vector Quantization* merupakan salah satu dari metode *Neural Network* yang ditemukan oleh T. Kohonen. Kelebihan LVQ salah satunya ialah kemampuannya dalam melakukan pelatihan pada lapisan yang kompetitif.

Penelitian yang terkait dengan penggunaan *Learning Vector Quantization* telah banyak dilakukan. Salah satu penelitian yang telah dilakukan oleh Meri Azmi (2014) didapatkan bahwa kedua metode tersebut dapat menunjukkan hasil perhitungan yang baik dimana akurasi yang didapatkan dengan metode LVQ adalah 86,66%. Meskipun metode LVQ mampu menyelesaikan permasalahan tersebut dengan hasil yang baik, namun hasil akurasi LVQ masih belum maksimal. Kendala dalam metode LVQ ialah penentuan bobot yang digunakan pada awal perhitungan yang dapat mempengaruhi hasil akurasinya. Sehingga diperlukan metode lain yang dapat digunakan untuk meningkatkan akurasi, salah satunya dalam penelitian yang dilakukan oleh Raissa Arniantya (2017). Dengan menggabungkan algoritme evolusi dan LVQ tersebut diperoleh hasil yang lebih baik dimana akurasi yang diperoleh mencapai 92%. Terdapat beberapa metode

dalam algoritme evolusi, diantaranya *Genetic Algorithm* dan *Particle Swarm Optimization.* Selain itu dalam penelitian yang dilakukan Haldar dan Mishra (2016), dibahas perihal penggunaan *Particle Swarm Optimization* (PSO) sebagai metode optimasi pada LVQ. Pada penelitian tersebut didapatkan hasil yang lebih baik pada LVQ yang dioptimasi dengan PSO dengan akurasi 92% dibandingkan dengan LVQ tanpa optimasi PSO dengan akurasi 90%.

Dari uraian diatas, maka penulis membuat penelitian dengan mengoptimasi algoritme *Learning Vector Quantization (LVQ)* dengan algoritme *Particle Swarm Optimization (PSO)* dimana algoritme *Particle Swarm Optimization (PSO)* memiliki kelebihan dari segi efisiensi dibanding dengan teknik optimasi lain. Sehingga penulis mengangkat judul "*Penerapan Algoritme Particle Swarm Optimization-Learning Vector Quantization (PSO-LVQ) Pada Klasifikasi Data Iris*" dengan menggunakan dataset iris yang diambil dari UCI Machine Learning yang dijadikan sebagai data latih dalam penelitian ini.

1.2 Rumusan masalah

Berdasarkan penjelasan dari latar belakang, maka rumusan masalah yang tercipta sebagai berikut:

1. Bagaimana penerapan algoritme *Particle Swarm Optimization-Learning Vector Quantization* (PSO-LVQ) untuk klasifikasi dataset Iris ?
2. Bagaimana tingkat akurasi dari algoritme *Particle Swarm Optimization-Learning Vector Quantization* (PSO-LVQ) untuk klasifikasi dataset Iris ?

1.3 Tujuan

Dari uraian rumusan masalah, tujuan yang diharapkan dalam penelitian ini meliputi:

1. Menerapkan algoritme *Particle Swarm Optimization-Learning Vector Quantization* (PSO-LVQ) untuk klasifikasi dataset Iris.
2. Menguji tingkat akurasi yang diperoleh algoritme *Particle Swarm Optimization-Learning Vector Quantization* (PSO-LVQ) untuk klasifikasi dataset Iris.

1.4 Manfaat

Manfaat yang nantinya akan diperoleh dari penelitian ini antara lain:

1. Bagi Peneliti
   - Sebagai media dalam implementasi ilmu pengetahuan teknologi pada bidang AI.
   - Menambah ilmu pengetahuan berupa penerapan algoritme *Particle Swarm Optimization-Learning Vector Quantization* (PSO-LVQ) untuk klasifikasi dataset Iris.

2. Bagi Masyarakat
   - Memberikan pengetahuan tentang perbedaan bunga Iris dan menambah ilmu tentang metode yang digunakan.

1.5 Batasan masalah

Batasan dari permasalahan penelitian ini meliputi:

1. Algoritme yang digunakan adalah *Particle Swarm Optimization-Learning Vector Quantization* (PSO-LVQ).
2. Data yang nantinya dipakai dalam skripsi ini berasal dari data *UCI Machine Learning (http://archive.ics.uci.edu/ml/datasets/iris),* menggunakan dataset Iris.
3. Atribut dari data Iris yang digunakan adalah *Sepal Length, Petal Length, Sepal Width* dan *Petal Width.*
4. Hasil klasifikasi yang diberikan adalah *Iris VersiColor, Iris Sentosa* dan *Iris Virginica.*

1.6 Sistematika pembahasan

1. **BAB I. PENDAHULUAN**
   Bab ini akan memuat perihal latar belakang, rumusan masalah, batasan masalah, tujuan dan manfaat dan sistematika dari penulisan.
2. **BAB II. LANDASAN KEPUSTAKAAN**
   Bab ini mengurai dasar-dasar teori yang berkaitan dengan masalah penulisan yang diangkat yang menjadi gambaran dalam membangun penelitian ini. Hal tersebut seperti Neural Network, klasifikasi, Iris, algoritme *Particle Swarm Optimization* (PSO) dan algoritme *Learning Vector Quantization* (LVQ).
3. **BAB III. METODOLOGI**
   Bab ini menjelaskan alur pengerjaan dalam menerapkan algoritme *Particle Swarm Optimization* (PSO) dan algoritme *Learning Vector Quantization* (LVQ) dalam penelitian ini.
4. **BAB IV. PERANCANGAN**
   Menjelaskan tentang semua perancangan dari klasifikasi menggunakan algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) pada dataset Iris.
5. **BAB V. IMPLEMENTASI**
   Menjelaskan pengimplementasian algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) pada klasifikasi dataset Iris.
6. **BAB VI. PENGUJIAN DAN ANALISIS**
   Menjelaskan alur dan hasil dari proses pengujian sistem yang telah dibuat dan sesuai dengan program dari perancangan yang disertai analisis.

7. **BAB Vll. PENUTUP**

   Memuat kesimpulan dari keseluruhan penjelasan bab-bab sebelumnya, serta saran-saran dari hasil yang diperoleh, yang diharapkan dapat bermanfaat dalam pengembangan selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini dijelaskan mengenai teori terkait Neural Network, klasifikasi, Iris, algoritme *Particle Swarm Optimization* (PSO) dan algoritme *Learning Vector Quantization* (LVQ). Penelitian terkait implementasi dari algoritme *Learning Vector Quantization* (LVQ) akan diuraikan kedalam kajian pustaka.

2.1 Kajian Pustaka

Kajian pustaka penelitian ini mengacu dari penelitian-penelitian sebelumnya dengan judul "*Komparasi Metode jaringan syaraf tiruan SOM dan LVQ untuk mengidentifikasi data bunga Iris"* yang dilakukan oleh Meri Azmi (2014). Pada penelitian tersebut dilakuan perbandingan algoritme LVQ dan juga algoritme SOM dengan menggunakan dataset Iris yang didapat dari *UCI Machine Learning*. Penelitian ini menghasilkan akurasi pada SOM sebesar 89.33% dan akurasi LVQ sebesar 86.67% dengan menggunakan bobot pertama pada setiap kelas pada dataset, yang menyimpulkan bahwa kedua algoritme tersebut cukup baik dalam menyelesaikan permasalahan tersebut namun dengan hasil LVQ berada dibawah akurasi SOM, selain itu nilai bobot awal juga sangat mempengaruhi hasil akhir akurasi.

Penelitian selanjutnya yang berjudul "*Optimasi vector bobot pada learning vector quantization menggunakan algoritme genetika untuk identifikasi jenis attention deficit hyperactivity disorder pada anak"* yang dilakukan oleh Raissa Arniantya (2017). Pada penelitian tersebut dilakukan perbandingan terhadap algoritme LVQ dan hybrid algoritme LVQ dengan *Genetic Algorithm* dengan data sebanyak 75, nilai *crossover 0.6,* nilai *mutation 0.4* dan juga iterasi sebanyak 1000 menghasilkan akurasi pada LVQ sebesar 77% dan LVQ-AG 92%. Dari penelitian ini didapatkan bahwa penggunaan hybrid dengan algoritme lain dapat meningkatkan tingkat akurasi dari LVQ.

Penenlitian lainnya yang berjudul *"An Approach for Iris plant classification using neural network"* yang dilakukan oleh Swain dkk(2012). Pada penelitian dengan menggunakan algoritme *Backpropagation* didapatkan hasil akurasi 83.33% pada iterasi 500 dan 96.66% pada iterasi 50000. Dari penelitian ini didapatkan bahwasannya semakin besar nilai iterasi dapat mempengaruhi peningkatan akurasi.

Penelitian selanjutnya yang berjudul *"Learning Vector Quantization (LVQ) Neural Network Approach for Multilingual Speech Recognition"* yang dilakukan oleh Rajat Haldar dan Pankaj Kumar (2016). Pada penelitian tersebut dilakukan perbandingan penggunaan algoritme *Learning Vector Quantization (LVQ)* dalam identifikasi *Speech Recognition* menggunakan *Particle Swarm Optimization* (PSO) dalam proses optimisasi data berbahasa inggris dan hindi dengan tanpa menggunakan *Particle Swarm Optimization* (PSO). Hasil yang diperoleh menunjukkan bahwa akurasi tanpa PSO sebesar 90% dan akurasi dengan PSO sebesar 92%.

Perbedaan pada penelitian ini dengan penelitian sebelumnya adalah algoritme *Learning Vector Quantization (*LVQ*)* yang di-*hybrid* dengan algoritme *Particle Swarm Optimization* (PSO) untuk mengatasi kelemahan pembobotan awal pada algoritme *Learning Vector Quantization* sehingga dapat meningkatkan hasil nilai akurasi yang optimal pada klasifikasi data Iris.

Tabel 2.1 Kajian Pustaka

No	Judul	Metode	Hasil
1	*Komparasi Metode jaringan syaraf tiruan SOM dan LVQ untuk mengidentifikasi data bunga Iris*	*Learning Vector Quantization (LVQ)* dan *Self Organizing Map (SOM)*	Memberikan perbadingan kedua metode dengan akurasi pada LVQ 86.67% dan SOM 89.33%, namun nilai bobot awal tetap.
2	*Optimasi vector bobot pada learning vector quantization menggunakan algoritme genetika untuk identifikasi jenis attention deficit hyperactivity disorder pada anak*	*Learning Vector Quantization (LVQ)* dan *Learning Vector Quantization-Genetic Algorithm (LVQ-GA)*	Nilai akurasi pada algoritme LVQ yang semula 77% mengalami peningkatan berkat hybrid GA menjadi 92%
3	*An Approach for Iris plant classification using neural network*	*Backpropagation*	Nilai kurasi pada metode ini adalah 83.33% pada iterasi 500 dan 96.66% pada iterasi 50000 dengan menggunakan 75 dataset.

4	*Learning Vector Quantization (LVQ) Neural Network Approach for Multilingual Speech Recognition*	*Learning Vector Quantization (LVQ)* dan *Particle Swarm Optimization* (PSO)	Akurasi tanpa PSO sebesar 90% dan akurasi dengan PSO sebesar 92%

2.2 Bunga Iris

Iris berasal dari Yunani yang berarti pelangi dan dikaitkan dengan dewi pelangi yang ada di Yunani, hal tersebut dibuktikan dengan banyaknya spesies yang dimiliki bunga Iris yang mencapai 300 macam spesies (Kamenetsky dan Hiroshi, 2013). Iris merupakan spesies tumbuhan yang saat ini telah banyak kita jumpai di berbagai tempat di seluruh dunia. Pada awalnya Iris hanya tumbuh di Eropa, Asia Barat, Afrika bagian utara, dan daerah medeterania.

Gambar 2.1 Bunga Iris VersiColor.

Sumber : Danielle Langlois di Forillon National Park (2015)

2.2.1 Dataset Iris

Data penelitian ini berasal dari dataset yang didapatkan pada *UCI Machine Learning.* Dataset ini berisikan data sebanyak 150 dengan jumlah kelas sebanyak 3 yaitu *Iris VersiColor, Iris Sentosa* dan *Iris Virginica.* Selain itu terdapat 4 atribut pada setiap data yaitu *Sepal Length, Petal Length, Sepal Width* dan *Petal Width.* Sepal adalah daun kelopak dari bunga Iris, sedangkan petal adalah daun mahkota dari bunga Iris.

2.3 Klasifikasi

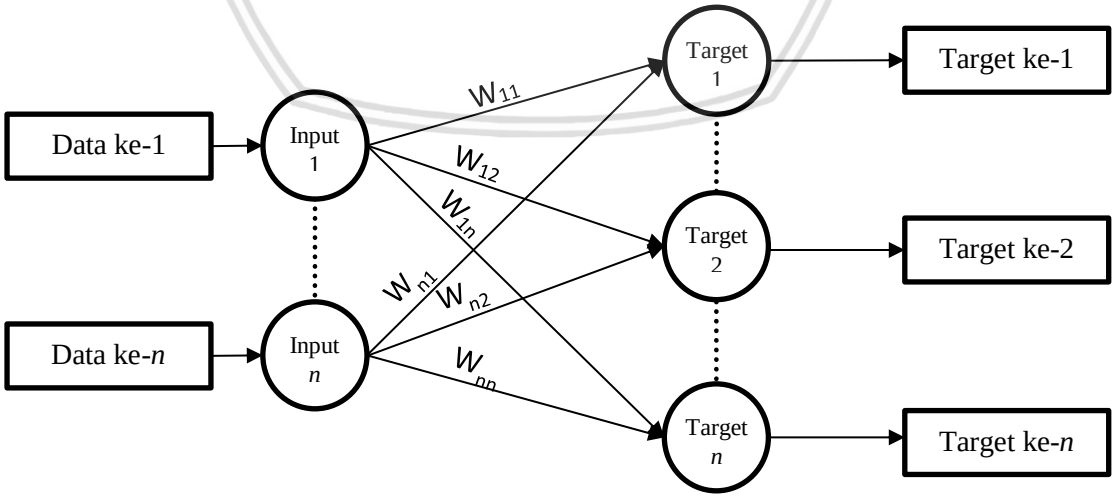
Klasifikasi dibagi dalam 2 jenis yaitu *Supervised Learning* dan *Unsupervised Learning. Supervised Learning*, yaitu belajar di mana satu set data latih akan diidentifikasi sesuai dengan kelompoknya. *Unsupervised Learning* atau yang dikenal sebagai pengelompokan, melibatkan pengelompokkan data ke dalam kategori berdasarkan beberapa ukuran kemiripan atau jarak yang melekat (Alpaydin, 2010).

2.3.1 Neural Network

Artificial Neural Network (ANN) atau Jaringan syaraf tiruan (JST) adalah representasi buatan otak manusia yang mencoba untuk meniru proses pembelajaran dari otak manusia. Kata buatan digunakan karena jaringan syaraf ini diterapkan dengan menggunakan program dari komputer yang diharapkan mampu menyelesaikan sejumlah proses perhitungan selama proses pembelajaran berjalan. Terdapat hal dari JST yang menyerupai cara kerja otak manusia, yaitu pengetahuan yang diperoleh dari proses belajar dan kekuatan hubungan antar sel syaraf (*neuron*) yang dikenal sebagai bobot-bobot yang digunakan untuk menyimpan pengetahuan. Kemampuan belajar tersebut dapat dianalogikan seperti sebuah proses manusia belajar untuk mengenali suatu hal (Wulandari, 2012).

2.4 *Learning Vector Quantization* (LVQ)

Learning Vector Quantization (LVQ) merupakan metode pelatihan untuk melakukan proses pembelajaran pada lapisan kompetitif yang terawasi (*supervised learning*) yang arsitekturnya berlayer tunggal (*single layer*). Kelas-kelas yang diperoleh sebagai sebuah hasil dari lapisan kompetitif ini bergantung pada jarak antar vektor-vektor yang menjadi input. Jika dua vektor input tersebut memiliki jarak yang mendekati sama, maka lapisan kompetitif akan berada pada kelas yang sama. LVQ merupakan metode klasifikasi pola yang mewakili kategori atau kelas tertentu (beberapa unit dari keluaran seharusnya digunakan untuk setiap kelas). Keunggulan metode ini adalah kemampuannya dalam memberikan pelatihan pada lapisan-lapisan kompetitif sehingga dapat mengklasifikasikan vektor input yang diberikan (Nugroho, 2011). LVQ memiliki beberapa variasi yang dapat digunakan yaitu LVQ1, LVQ2, LVQ2.1 dan LVQ3 (Fausett, 1994). Pada penelitian ini digunakan algoritme LVQ2 dalam proses *learning* nantinya.



Gambar 2.2 Arsitektur LVQ

Langkah-langkah dari algoritme LVQ terdiri atas (Wuryandari, 2012):

1. Inisialisasi

   a. Inisialisasi bobot awal

      Inisialisasi untuk bobot awal pada algoritme *Learning Vector Quantization* diawali dengan memilih data secara acak dari data yang ada. Bobot awal ini nantinya akan menjadi bobot acuan dalam proses pelatihan dengan algoritme ini. Sehingga diperoleh Persamaan 2.1.

      $$\boldsymbol{W_i = Rand[N_i]} \quad \textbf{(2.1)}$$

      Keterangan:

      W_i = bobot kelas ke-*i.*

      N_i = data kelas ke-*i* .

2. Proses pelatihan

   a. Lakukan perulangan sebanyak *epoch* yang telah ditentukan dan lakukan *update* pada *epoch* apabila telah dilakukan pelatihan terhadap data latih yang disediakan. Proses *update epoch* dilakukan terus menerus hingga mencapai batas maksimal *epoch* yang telah ditentukan.

      $$\boldsymbol{epoch = epoch + 1} \quad \textbf{(2.2)}$$

   b. Lakukan proses pelatihan terhadap seluruh data latih yang ada. Proses ini dilakukan di dalam proses perulangan *epoch.* Proses pelatihan bobot menggunakan Persamaan 2.3 dengan data latih yang disediakan. Perhitungan jarak sendiri menggunakan rumus *euclidean distance.*

      $$\boldsymbol{D_i = \sqrt{\sum_{j=1}^{n}(X_j - W_{ij})^2}} \quad \textbf{(2.3)}$$

      Keterangan:

      D_i = Jarak antara bobot dan data pada kelas-*i.*

      n = Jumlah seluruh parameter yang ada.

      X_j = Parameter data latih ke-*j.*

      W_{ij} = Bobot pada kelas ke-*i* dan parameter ke-*j.*

   c. Cari jarak terkecil dari proses perhitungan jarak yang telah dilakukan sehingga didapatkan jarak terkecil pertama dan kedua dengan Persamaan 2.4 dan Persamaan 2.5.

      $\boldsymbol{D_c}$ = jarak terkecil pertama, dimana:

      $$\boldsymbol{c = \min_i(D_i)} \quad \textbf{(2.4)}$$

      $$\boldsymbol{D_r = \min_i(D_i)}, \text{ dimana } \boldsymbol{i \neq c} \quad \textbf{(2.5)}$$

Keterangan:

D_c = Jarak terkecil pertama.

D_r = Jarak terkecil kedua.

$Min(D_i)$ = Nilai terkecil ke-*i*.

d. Lakukan *update* terhadap bobot yang sesuai jarak terkecil pertama dan kedua dengan ketentuan:

- Kelas dari data latih ke-*i* sama dengan kelas bobot dari perhitungan jarak terkecil kedua (D_r).
- Jarak terkecil pertama dan kedua memenuhi persyaratan *window* pada Persamaan 2.6.

$$\frac{D_c}{D_r} > 1 - \varepsilon \quad \text{dan} \quad \frac{D_r}{D_c} < 1 + \varepsilon \text{ , dimana } \varepsilon = 0.35 \qquad \textbf{(2.6)}$$

Keterangan:

D_c = Jarak terkecil pertama.

D_r = Jarak terkecil kedua.

ε = nilai *epsilon*.

Persamaan tersebut digunakan untuk mengetahui apakah jarak data latih dengan bobot berada didalam wilayah window ataupun tidak. *Window* merupakan wilayah antara jarak bobot terkecil pertama dan bobot terkecil kedua dengan data latih. Apabila telah memenuhi persyaratan dari *window* maka bobot akan dilakukan proses *update* secara simultan dengan Persamaan 2.7 dan Persamaan 2.8.

$$\mathbf{W_c(t+1) = W_c(t) - \alpha\,(t)[X_i(t) - W_c(t)]} \qquad \textbf{(2.7)}$$

$$\mathbf{W_r(t+1) = W_r(t) + \alpha\,(t)[X_i(t) - W_r(t)]} \qquad \textbf{(2.8)}$$

Keterangan:

$W_c(t+1)$ = Bobot dari jarak terkecil pertama baru.

$W_r(t+1)$ = Bobot dari jarak terkecil kedua baru.

$W_c(t)$ = Bobot dari jarak terkecil pertama saat ini.

$W_r(t)$ = Bobot dari jarak terkecil kedua saat ini.

$\alpha(t)$ = Nilai alpha saat ini.

$X_i(t)$ = Bobot data latih pada kelas ke-*i* saat ini.

e. Apabila tidak memenuhi persyaratan yang ada maka dilakuan *update* pada bobot dengan jarak terkecil pertama dengan persamaan berikut:

- Bila jarak terkecil pertama (D_c) memiliki kelas yang sama dengan data latih maka *update* dengan Persamaan 2.9.

$$\mathbf{W_c(t+1) = W_c(t) + \alpha\,(t)[X_i(t) - W_c(t)]} \quad \textbf{(2.9)}$$

Keterangan:

$W_c(t+1)$ = Bobot dari jarak terkecil pertama baru.

$W_c(t)$ = Bobot dari jarak terkecil pertama saat ini.

$\alpha(t)$ = Nilai alpha saat ini.

$X_i(t)$ = Bobot data latih pada kelas ke-*i* saat ini.

- Apabila jarak terkecil pertama (D_c) tidak memiliki kelas yang sama dengan data latih maka bobot akan dilakukan proses *update* dengan Persamaan 2.7.

f. Apabila telah melakukan pelatihan terhadap seluruh data latih, maka selanjutnya dilakukan proses *update* pada nilai *alpha* dengan persamaan 2.10.

$$\boldsymbol{\alpha(t+1) = \alpha(init) * [1 - (i/m)]} \quad \textbf{(2.10)}$$

Keterangan:

$\alpha(t+1)$ = nilai *alpha* baru.

$\alpha(init)$ = nilai *alpha* awal.

i = iterasi saat ini.

m = iterasi maksimum.

2.5 *Particle Swarm Optimization* (PSO)

Particle Swarm Optimization (PSO) merupakan metode optimasi yang diperkenalkan Kennedy dan Eberhart tahun 1995 berdasarkan penelitian dengan melihat perilaku kawanan burung. Setiap partikel pada *Particle Swarm Optimization* memiliki kecepatan partikel yang dinamis sesuai dengan perilaku historis mereka. Maka dari itu, partikel berkecenderungan untuk bergerak menuju daerah pencarian yang lebih baik selama proses pencarian. Dalam algoritme PSO terdapat beberapa proses sebagai berikut:

1. Inisialisasi

   a. Tentukan batas maksimum dan minimum dari partikel dengan melihat nilai maksimum dan minimum pada data latih yang digunakan dengan persamaan berikut:

$$\boldsymbol{T_{j,max} = max\{N_j\}} \quad \textbf{(2.11)}$$

$$\boldsymbol{T_{j,min} = min\{N_j\}} \quad \textbf{(2.12)}$$

Keterangan:

$T_{j,max}$ = Batas maksimum partikel pada parameter ke-*j*.

$T_{j,min}$ = Batas minimum partikel pada parameter ke-*j*.

N_j = Parameter data latih ke-*j.*

b. Selanjutnya tentukan batas maksimum dan minimun dari kecepatan dengan Persamaan 2.13 dan Persamaan 2.14.

$$\boldsymbol{V_{j,max} = \frac{1}{2}(T_{j,max} - T_{j,min})} \quad \textbf{(2.13)}$$

$$\boldsymbol{V_{j,min} = -\frac{1}{2}(T_{j,max} - T_{j,min})} \quad \textbf{(2.14)}$$

Keterangan:

$T_{j,max}$ = Batas maksimum partikel pada parameter ke-*j*.

$T_{j,min}$ = Batas minimum partikel pada parameter ke-*j*.

$V_{j,min}$ dan $V_{j,max}$ = Batas kecepatan pada parameter ke-*j*.

c. Inisialisasi kecepatan awal

Pada iterasi ke-0, dapat dipastikan bahwa nilai kecepatan awal semua partikel adalah 0.

$$\boldsymbol{V_{ij} = 0} \quad \textbf{(2.15)}$$

Keterangan:

V_{ij} = Kecepatan pada data ke-*i* dan parameter ke-*j*.

d. Inisialisasi posisi awal partikel

Pada iterasi ke-0, posisi awal partikel dibangkitkan dengan persamaan:

$$\boldsymbol{x_{ij}(0) = T_{j,min} + rand\,[0,1]\,.\,(T_{j,max} - T_{j,min})} \quad \textbf{(2.16)}$$

Keterangan:

x_{ij} = Partikel ke-*i* dan parameter ke-*j*.

$T_{j,max}$ = Batas maksimum partikel pada parameter ke-*j*.

$T_{j,min}$ = Batas minimum partikel pada parameter ke-*j*.

e. Inisialisasi *Pbest* dan *Gbest*

Ketika iterasi ke-0, nilai *Pbest* akan disamakan dengan nilai posisi awal dari partikel. Sedangkan nilai *Gbest* dipilih salah satu *Pbest* dengan *fitness* tertinggi.

$$\boldsymbol{Pbest_{ij}(0) = x_{ij}(0)} \quad \textbf{(2.17)}$$

$$\boldsymbol{Gbest(0) = Pbest_{k}(0)} \quad \textbf{(2.18)}$$

Keterangan:

$Pbest_{ij}(0)$ = Nilai partikel terbaik di iterasi 0 pada *Pbest* ke-*i* dan parameter ke-*j.*

x_{ij} = Partikel ke-*i* dan parameter ke-*j*.

$Gbest(0)$ = Nilai partikel terbaik keseluruhan di iterasi 0.

k = Posisi partikel dengan nilai *fitness* terbaik.

2. *Update* kecepatan

Untuk melakukan *update* kecepatan, digunakan rumus berikut:

$$\boldsymbol{v_{i,j}^{(t+1)} = w.v_{i,j}^{t} + c_1.r_1(Pbest_{i,j}^{t} - x_{i,j}^{t}) + c_2.r_2(Gbest_{j}^{t} - x_{i,j}^{t})}$$

$$v_{ij}^{t+1} = \begin{cases} v_{max}, & v_{ij}^{t+1} \geq v_{max} \\ v_{min}, & v_{ij}^{t+1} \leq v_{min} \\ v_{ij}^{t+1}, & v_{min} \geq v_{ij}^{t+1} \geq v_{max} \end{cases} \quad \textbf{(2.19)}$$

Keterangan:

$Pbest_{ij}(t)$ = Nilai *Pbest* di iterasi ke-*t* pada *Pbest* ke-*i* dan parameter ke-*j.*

x_{ij}^{t} = Partikel ke-*i* dan parameter ke-*j* pada iterasi ke-*t*.

$Gbest_{j}^{t}(0)$ = Nilai parameter Gbest ke-*j* di iterasi ke-*t*.

w = Bobot *inersi.*

v_{ij}^{t} = Kecepatan ke-*i* dan parameter ke-*j* pada iterasi ke-*t*.

c_1 dan c_2 = Nilai konstanta.

r_1 dan r_2 = Nilai acak dari distribusi *uniform* [0,1].

v_{min} = Batas minimum kecepatan.

v_{max} = Batas maksimum kecepatan.

3. *Update* posisi partikel dan menghitung *fitness*

Untuk proses *update* posisi, akan digunakan rumus berikut:

$$\boldsymbol{x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1}}$$

$$\boldsymbol{x_{ij}^{t+1} = \begin{cases} T_{j,max}, & x_{ij}^{t+1} \geq T_{j,max} \\ T_{j,min}, & x_{ij}^{t+1} \leq T_{j,min} \\ x_{ij}^{t+1}, & T_{j,min} \geq x_{ij}^{t+1} \geq T_{j,max} \end{cases}} \quad \textbf{(2.20)}$$

Keterangan:

x_{ij}^{t} = Partikel ke-*i* dan parameter ke-*j* pada iterasi ke-*t*.

x_{ij}^{t+1} = Partikel ke-*i* dan parameter ke-*j* pada iterasi ke-*t+1*.

v_{ij}^{t+1} = Kecepatan ke-*i* dan parameter ke-*j* pada iterasi ke-*t+1*.

$T_{j,max}$ = Batas maksimum partikel pada parameter ke-*j*.

$T_{j,min}$ = Batas minimum partikel pada parameter ke-*j*.

4. *Update* nilai Pbest dan nilai Gbest

Akan dilakukan perbandingan antara *fitness* pBest pada iterasi sebelumnya dengan hasil fitness dari *update* posisi. Fitness yang lebih tinggi akan menjadi pBest yang baru. pBest terbaru yang memiliki nilai fitness tertinggi akan menjadi gBest yang baru.

$$Pbest_i^{t+1} = \begin{cases} Pbest_i^t, & fitness(x_i^{t+1}) \leq fitness(Pbest_i^t) \\ x_i^{t+1}, & fitness(x_i^{t+1}) > fitness(Pbest_i^t) \end{cases}$$

(2.21)

$$Gbest^{t+1} = \begin{cases} Gbest^t, & argmax\{fitness(Pbest_i^{t+1})\} \leq fitness(Gbest^t) \\ Pbest_i^{t+1}, & argmax\{fitness(Pbest_i^{t+1})\} > fitness(Gbest^t) \end{cases}$$

(2.22)

Keterangan:

$Pbest_i^t$ = Nilai *fitness Pbest* di iterasi ke-*t* pada *Pbest* ke-*i*.

$Pbest_i^{t+1}$ = Nilai *fitness Pbest* di iterasi ke-*t+1* pada *Pbest* ke-*i*.

x_i^{t+1} = Nilai *fitness* partikel ke-*i* iterasi ke-*t+1*.

$Gbest_i^t$ = Nilai *fitness Gbest* di iterasi ke-*t* pada *Gbest* ke-*i*.

$Gbest_i^{t+1}$ = Nilai *fitness Gbest* di iterasi ke-*t+1* pada *Gbest* ke-*i*.

$argmax\{fitness(Pbest_i^{t+1})\}$ = Nilai *fitness Pbest* terbaik ke-*i* di iterasi ke-*t*.

5. ulangi langkah 2 – 5 hingga mencapai iterasi yang telah ditentukan.

2.6 Perhitungan Evaluasi

Perhitungan evaluasi ini digunakan pada proses perhitungan *fitness* pada algoritme *Particle Swarm Optimization* dan digunakan untuk proses evaluasi dari proses pelatihan pada algoritme *Learning Vector Quantization.* Perhitungan evaluasi sendiri memiliki tahapan beirkut:

1. Hitung jarak antara bobot/partikel dengan data yang digunakan sebagai data latih sehingga akan didapatkan jarak dari setiap bobot / partikel yang ada. Jarak tersebut nantinya akan dicari jarak terpendeknya dan akan dilihat apakah jarak terpendek tersebut berada di kelas yang sama dengan kelas dari data latih yang digunakan dalam perhitungan jarak. Perhitungan jarak sendiri menggunakan persamaan berikut:

$$\boldsymbol{D_i = \sqrt{\sum_{j=1}^{n}(X_j - W_{ij})^2}} \quad \textbf{(2.23)}$$

Keterangan:

D_i = Jarak antara bobot/partikel dan data pada kelas-*i.*

n = Jumlah seluruh parameter yang ada.

X_j = Parameter data latih ke-*j.*

W_{ij} = Bobot pada kelas ke-*i* dan parameter ke-*j.*

2. Selanjutnya hitung jumlah data latih yang memiliki kesamaan kelas antara bobot/partikel dengan kelas data latih berdasarkan jarak terkecil yang telah dihitung dengan Persaaan 2.23. perhitungan jumlah data ini menggunakan persamaan berikut:

$$\boldsymbol{Akurasi = \frac{c}{n}} \quad \textbf{(2.24)}$$

$$\boldsymbol{Akurasi = Akurasi * 100\%} \quad \textbf{(2.25)}$$

Keterangan:

c = Jumlah seluruh data uji yang sesuai.

n = Jumlah seluruh data uji.

2.7 Particle Swarm Optimization - Learning Vector Quantization (PSO-LVQ)

Particle Swarm Optimization-Learning Vector Quantization (PSO-LVQ) merupakan algoritme yang berasal dari algoritme *Learning Vector Quantization*

yang di *hybrid* dengan menggunakan algoritme *Particle Swarm Optimization*. Alur algoritme dapat dilihat pada tahapan berikut:

1. Perhitungan bobot dengan algoritme *Particle Swarm Optimization*
   - Pada tahap awal algoritme ini adalah inisialisasi data, inisialisasi data sendiri memiliki kesamaan dengan algoritme *Particle Swarm Optimization* yang telah dijabarkan pada Sub bab 2.5. Tahapan inisialisasi algoritme ini sebagai berikut:
     a. Inisialisasi batas maksimum dan minimum partikel dengan Persamaan 2.11 dan Persamaan 2.12.
     b. Inisialisasi batas maksimum dan minimum kecepatan dengan Persamaan 2.13 dan Persamaan 2.14.
     c. Inisialisasi nilai partikel awal dengan Persamaan 2.16.
     d. Hitung nilai *fitness* setiap partikel yang ada dengan Persamaan 2.23 dan Persamaan 2.24.
     e. Inisialisasi nilai awal kecepatan dengan Persamaan 2.15.
     f. Tentukan nilai Pbest dengan Persamaan 2.17.
     g. Tentukan nilai Gbest berdasarkan nilai *fitness* dengan Persamaan 2.18.
   - Selanjutnya lakukan *update* terhadap kecepatan dengan Persamaan 2.19.
   - Lakukan *update* terhadap nilai partikel, nilai *Pbest*, dan *Gbest* dengan persamaan 2.20, Persamaan 2.21, dan Persamaan 2.22.
   - Ulangi tahap *update* hingga mencapai iterasi yang telah ditentukan.

   Dari tahapan diatas nantinya akan didapatkan bobot terbaik yang didapat dari Gbest yang merupakan hasil perhitungan dengan algoritme *Paricle Swarm Optimization.* Bobot tersebut nantinya akan digunakan sebagai bobot awal pada perhitungan algoritme *Learning Vector Quantization.*

2. Pelatihan dengan algoritme *Learning Vector Quantization*

   Pada proses pelatihan menggunakan algoritme *Learning Vector Quantization* dilakukan dengan beberapa tahapan yang mempunyai kesamaan dengan tahapan pada Sub bab 2.4. Tahapan algoritme *Learning Vector Quantization* sebagai berikut:

   - Inisialisasi bobot awal pada algoritme ini menggunakan nilai Gbest terakhir pada proses algoritme *Particle Swarm Optimization*.
   - Tahap pelatihan pada proses ini memiliki tahapan yang sama dengan algoritme *Learning Vector Quantization* yang telah dijabarkan sebelumnya dengan menggunakan Persamaan 2.2, Persamaan 2.3,

Persamaan 2.4, Persamaan 2.5, Persamaan 2.6, Persamaan 2.7, Persamaan 2.8, Persamaan 2.9, dan Persamaan 2.10.

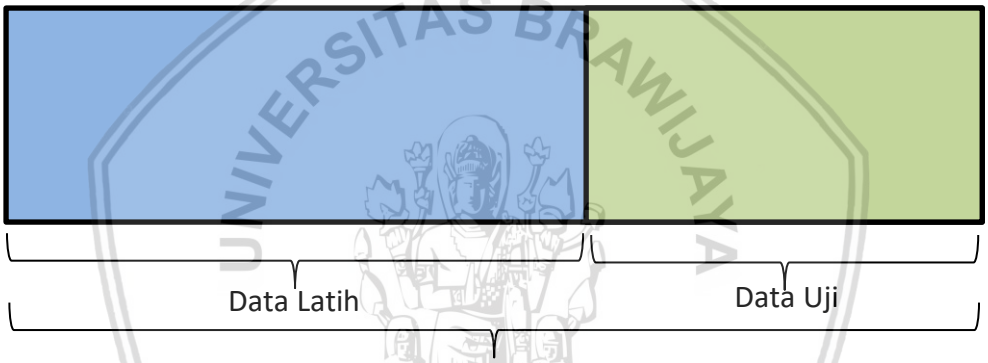
3. Tahap Evaluasi

Pada proses ini dilakukan perhitungan untuk mengetahui bagaimana hasil dari algoritme ini, untuk proses perhitungan evaluasi ini menggunakan Persamaan 2.23 dan Persamaan 2.24.

2.8 Cross Validation

Cross Validation merupakan salah satu teknik dalam pengujian suatu data. Pengujian tersebut dilakukan dengan membagi dua bagian data yang ada menjadi data uji dan juga data latih. *Cross validation* memiliki beberapa tipe seperti *Holdout Method, K-Fold, Leave-P-Out* dan *Bootstrap*.

2.8.1 Holdout Method

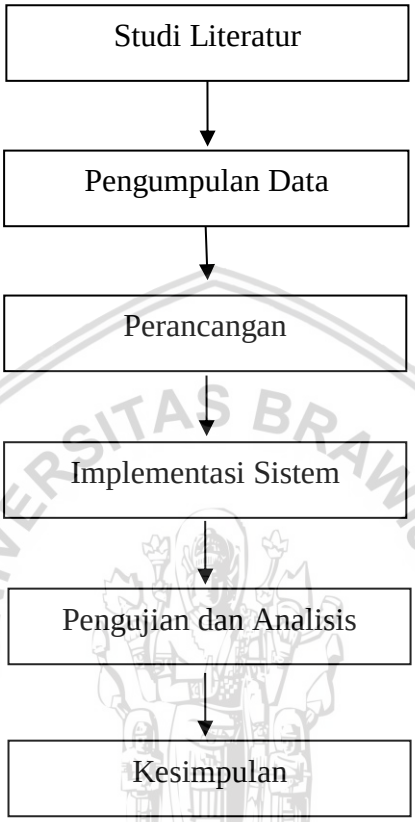


Gambar 2.3 Pembagian Data *Holdout Method*

Holdout Method merupakan salah satu dari beberapa metode yang ada dalam Cross Validation. *Holdout Validation* merupakan metode yang sederhana dalam proses validasi data, dimana data yang ada akan dibagi menjadi dua bagian yaitu data latih dan juga data uji dimana data uji memiliki ukuran yang lebih kecil dibandingkan data latih. Data latih merupakan data yang nantinya akan digunakan dalam proses pelatihan / *training* data, setelah melakukan proses *training* data barulah data akan diuji menggunakan data uji yang sebelumnya telah ditentukan.

BAB 3 METODOLOGI

Metodologi penelitian yang dikerjakan pada penelitian ini dilalui dalam tahapan-tahapan yang akan digambarkan dengan gambaran diagram blok metodologi penelitian berikut :



Gambar 3.1 Blok diagram metodologi penelitian

3.1 Studi Pustaka

Metode yang nantinya digunakan dalam memperoleh informasi tambahan yang berguna sebagai acuan. Mempelajari literatur yang berasal dari beberapa bidang ilmu yang saling terikat dengan Penerapan Algoritme *Particle Swarm Optimization-Learning vector quantization (PSO-LVQ)* pada klasifikasi data Iris, diantaranya :

- Iris
- *Klasifikasi*
- *Neural Network*
- *Learning Vector Quantization*
- *Particle Swarm Optimization*

Literatur tersebut didapatkan dari jurnal-jurnal, buku pengetahuan, artikel ataupun dokumentasi dari sebuah proyek.

3.2 Pengumpulan Data

Berdasarkan Batasan masalah yang telah disebutkan sebelumnya, data diambil dari website *UCI Machine Learning (https://archive.ics.uci.edu/ml/datasets/iris).* Dataset yang diambil berupa 150 data yang terdiri atas 3 kelas yaitu *Iris Sentosa, Iris VersiColor* dan *Iris Virginica*. Selain itu dataset terdiri atas 4 atribut yaitu *Sepal Length, Sepal Width, Petal Length* dan *Petal Width.*

3.3 Perancangan

Pada tahap perancangan ini, akan dijelaskan permasalahan yang akan dibahas dan juga mengenai alur algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) yang digunakan nantinya, selain itu pada tahap ini akan berisikan perhitungan manual dari *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) yang digunakan pada penelitian ini.

3.4 Implementasi

Implementasi dilakukan dengan menggunakan bahasa pemrograman *Java*. Implementasi nantinya akan menerapkan algoritme algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ). Selain itu, implementasi juga dilakukan dengan mengikuti perhitungan manual untuk klasifikasi data Iris dengan algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ).

3.5 Analisis dan Pengujian

Analisis yang dikerjakan dalam menentukan nilai akurasi dari data yang telah dibandingkan dan hasil data tersebut diolah dalam perhitungan manual dengan perhitungan yang dikerjakan oleh sistem untuk penerapan algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) pada klasifikasi data Iris.

3.6 Kesimpulan

Kesimpulan akan diambil ketika semua alur tahapan telah selesai dilakukan dari tahap-tahap sebelumnya yang sudah dilakukan. Tahap terakhir adalah memberi beberapa saran dengan tujuan mengurangi kesalahan yang ada serta memberikan pilihan dalam pengembangan kedepannya.

BAB 4 PERANCANGAN

Pada bab perancangan ini, akan mambahas perihal permasalahan bunga Iris yang akan diklasifikasi degan menggunakan algoritme PSO-LVQ, selain itu bab ini akan membahas bagaimana siklus algoritme yang akan digunakan beserta perhitungan secara manual.

4.1 Deskripsi Masalah

Masalah yang akan diselesaikan nantinya adalah klasifikasi data bunga Iris yang memiliki empat atribut yaitu *sepal length, petal length, sepal width* dan *petal width.* Selain itu data Iris ini memiliki tiga macam kelas yaitu *Iris versicolor, Iris-setosa* dan *Iris-virginica.*

Data tersebut berisikan ukuran dari *sepal* (helai kelopak) dan *petal* (helai mahkota) dari tiga macam kelas yang tersedia. Dari data tersebut nantinya akan dilakukan proses klasifikasi dengan algoritme *Particle Swarm Optimization-Learning Vector Quantization (*PSO-LVQ). Dari algoritme tersebut nantinya akan terbagi dua tahapan, tahap pertama akan dilakukan proses untuk mencari bobot terbaik dengan *Particle Swarm Optimization* berdasarkan data yang tersedia dan bobot hasil proses *Particle Swarm Optimization* nantinya akan digunakan pada *Learning Vector Quantization* sebagai bobot awal dari proses pelatihan.

Adapun data yang nantinya digunakan dalam proses perhitungan secara manual pada bab ini ada pada Tabel 4.1 dimana data tersebut berjumlah 6 data yang terdiri dari tujuh data yang berasal dari setiap kelas yang ada.

Tabel 4.1 Data Bunga Iris

No	*Sepal Length*	*Sepal Width*	*Petal Length*	*Petal Width*	Kelas	
1.	5,1	3,5	1,4	0,2	*Iris-setosa*	1
2.	4,9	3	1,4	0,2	*Iris-setosa*	1
3.	7	3,2	4,7	1,4	*Iris-versicolor*	2
4.	6,4	3,2	4,5	1,5	*Iris-versicolor*	2
5.	6,3	3,3	6	2,5	*Iris-virginica*	3
6.	5,8	2,7	5,1	1,9	*Iris-virginica*	3

4.2 Siklus Algoritme

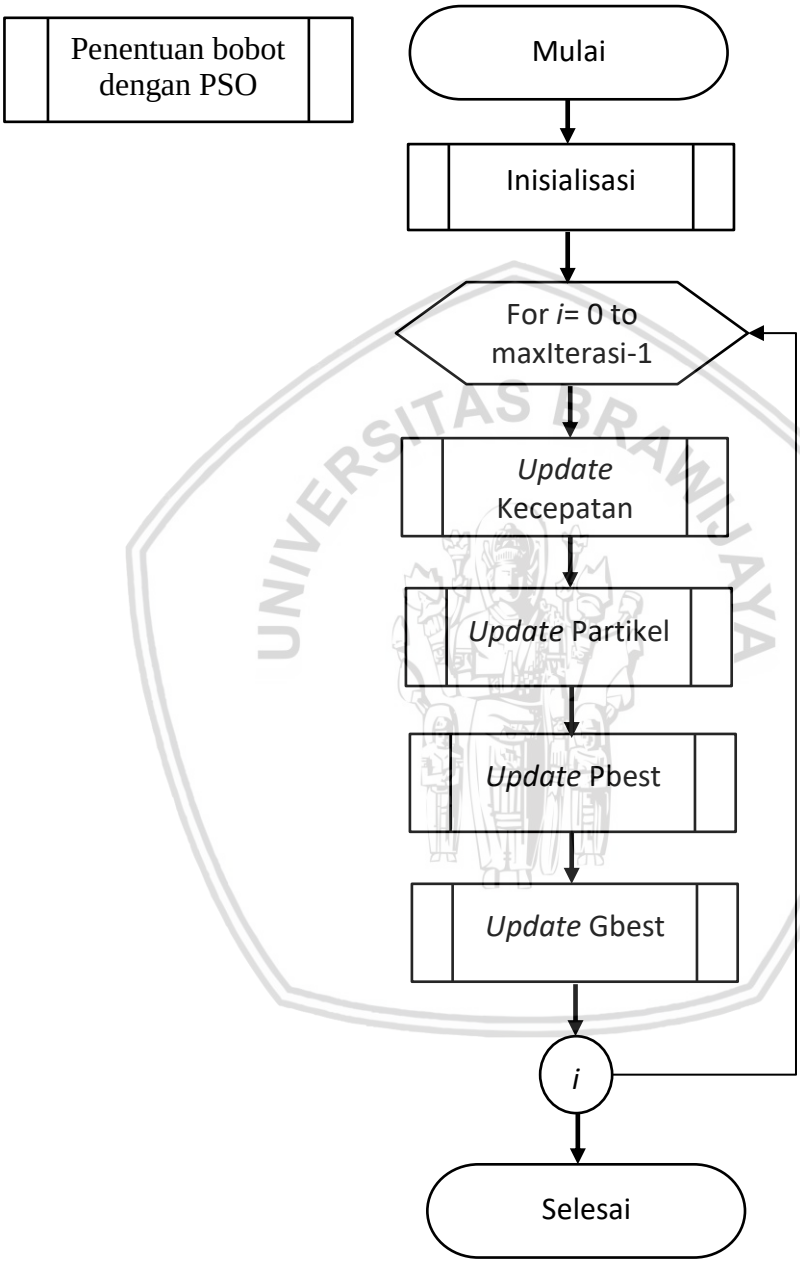
Pada bagian ini akan menjelaskan siklus algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) dalam melakukan proses klasifikasi pada data Iris. Gambar 4.1 menjelaskan proses kerja algoritme secara keseluruhan.

Gambar 4.1 Alur Algoritme dari PSO - LVQ

1. Membaca masukan data latih.
2. Melakukan proses perhitungan untuk mendapatkan bobot terbaik dengan PSO.
3. Melakukan proses pengklasifikasian data dengan LVQ.
4. Melakukan pengujian untuk mengetahui tingkat akurasi dari keseluruhan proses.
5. Menampilkan hasil dari pengujian yang telah dilakukan.

4.2.1 Proses *Particle Swarm Optimization*

Pada proses *Particle Swarm Optimization* ini akan dilakukan pencarian bobot terbaik yang nantinya akan menjadi bobot awal dalam proses pengklasifikasian dengan *Learning Vector Quantization*. Gambar 4.2 menjelaskan proses dari algoritme PSO ini.



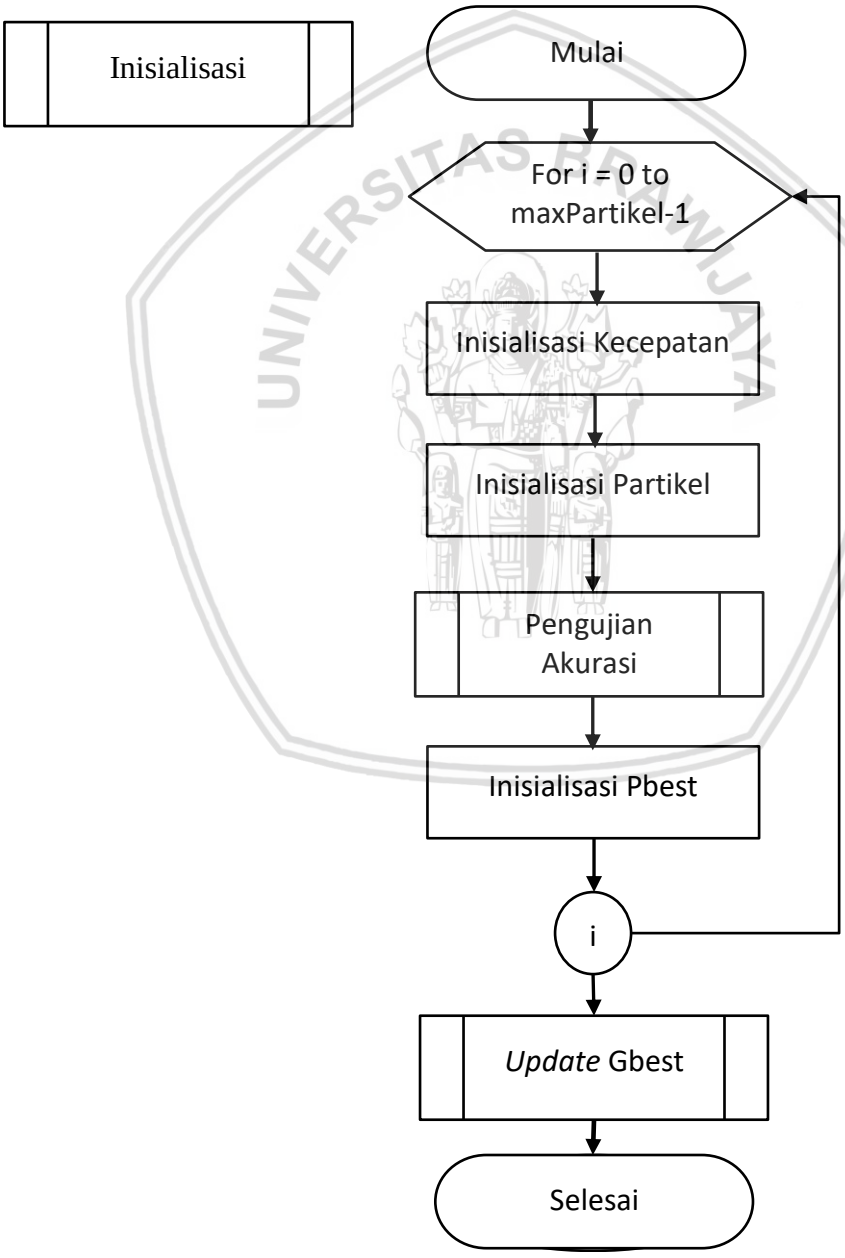
Gambar 4.2 Alur algoritme *Particle Swarm Optimization*

1. Lakukan proses inisialisasi data.
2. Lakukan perulangan hingga tidak memenuhi syarat iterasi.

3. *Update* kecepatan yang sebelumnya telah diberi nilai dengan Persamaan 2.19.
4. *Update* partikel yang sudah ada dengan kecepatan yang telah diperbaharui menggunakan Persamaan 2.20.
5. *Update* nilai *Pbest* menggunakan Persamaan 2.21.
6. *Update* nilai *Gbest* menggunakan Persamaan 2.22.

4.2.1.1 Proses Inisialisasi

Berikut ini merupakan bagian dari alur algoritme *Particle Swarm Optimization* pada sub proses inisialisasi data. Gambar 4.3 menjelaskan alur dari proses inisialisasi data.

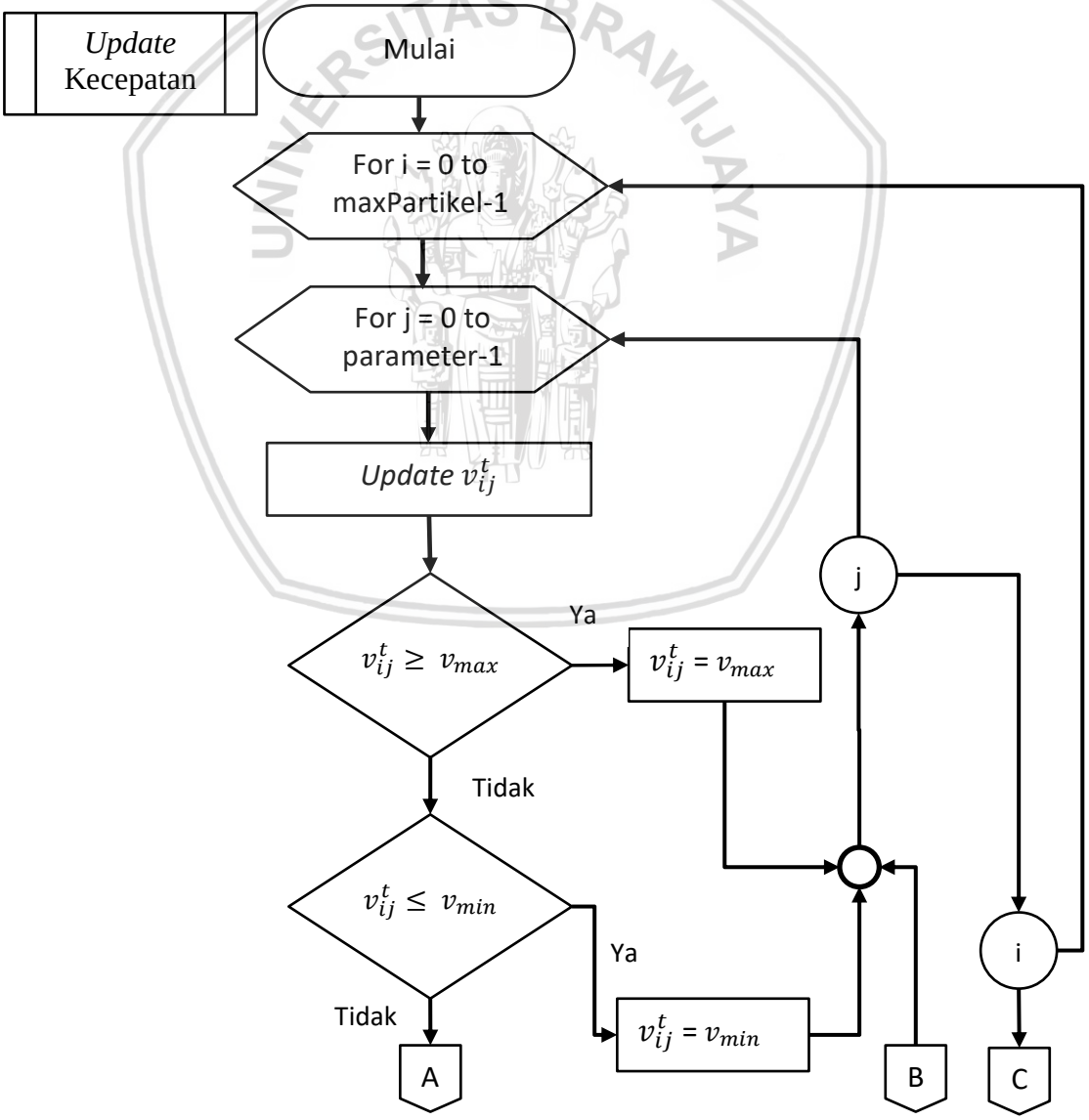


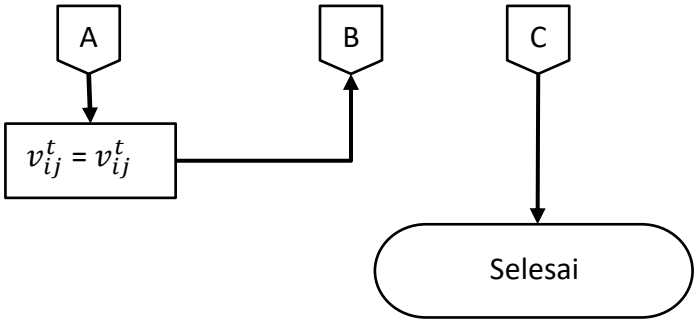
Gambar 4.3 Alur Inisialisasi pada Particle Swarm Optimization

1. Lakukan perulangan sebanyak partikel yang digunakan.
2. Melakukan inisialisasi kecepatan dengan Persamaan 2.15.
3. Melakukan inisialisasi partikel dengan persamaan 2.16.
4. Menjalankan fungsi pengujian akurasi untuk mendapatkan nilai *fitness* pada setiap partikel.
5. Inisialisasi nilai pbest dengan persamaan 2.17.
6. Jalankan fungsi *Update* gbest untuk memperoleh nilai dari Gbest.

4.2.1.2 Proses Update Kecepatan

Berikut ini merupakan bagian dari alur algoritme *Particle Swarm Optimization* pada sub proses *update* kecepatan. Gambar 4.4 menjelaskan alur dari proses *update* kecepatan.



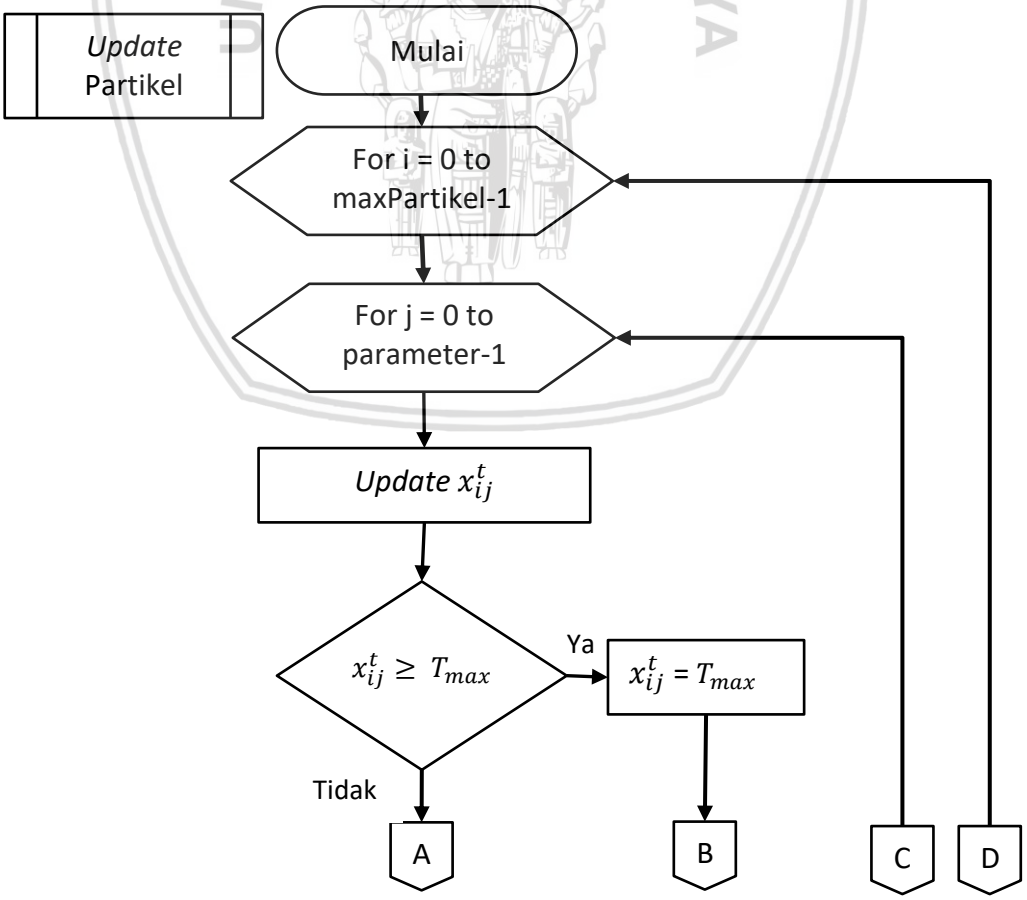


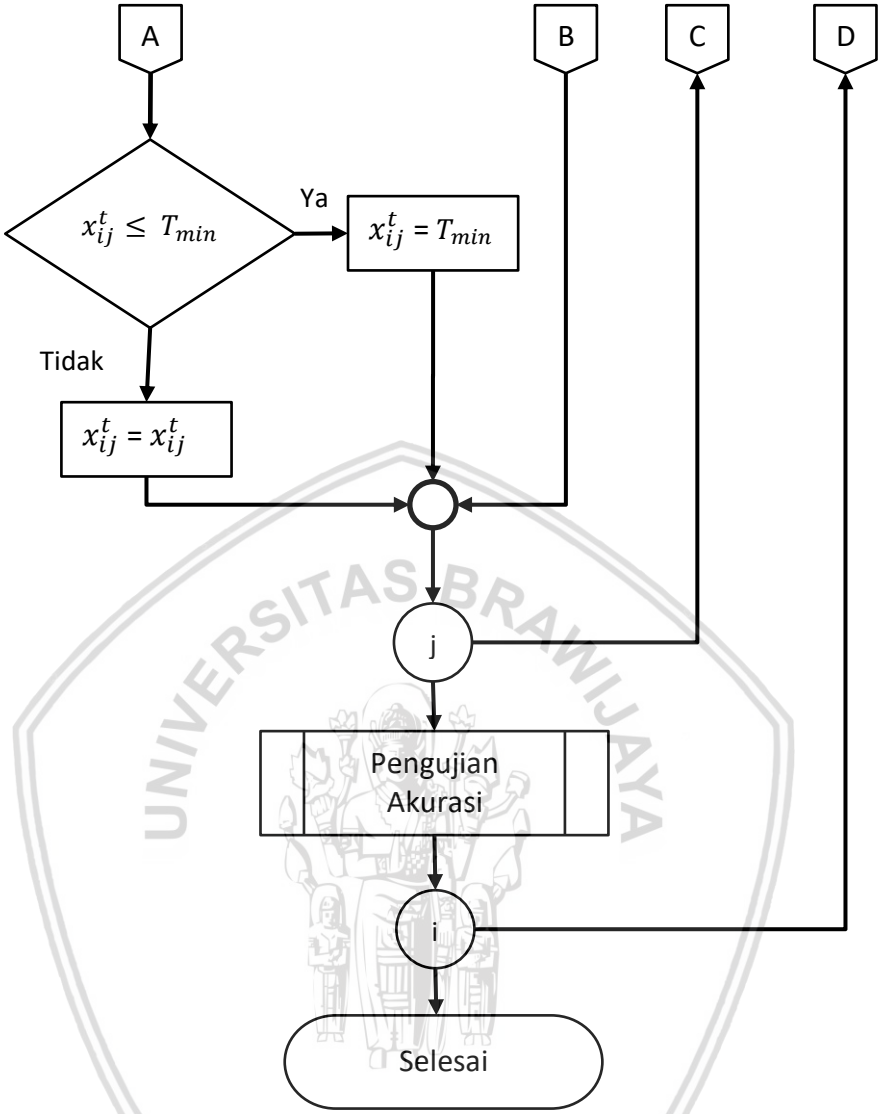
Gambar 4.4 Proses *Update* Kecepatan

1. Lakukan perulangan sebanyak partikel yang ada.
2. Lakukan perulangan sebanyak parameter yang ada pada partikel.
3. Lakukan *update* kecepatan dengan Persamaan 2.19.

4.2.1.3 Proses Update Partikel

Berikut ini merupakan bagian dari alur algoritme *Particle Swarm Optimization* pada sub proses *update* partikel. Gambar 4.5 menjelaskan alur dari proses *update* partikel.



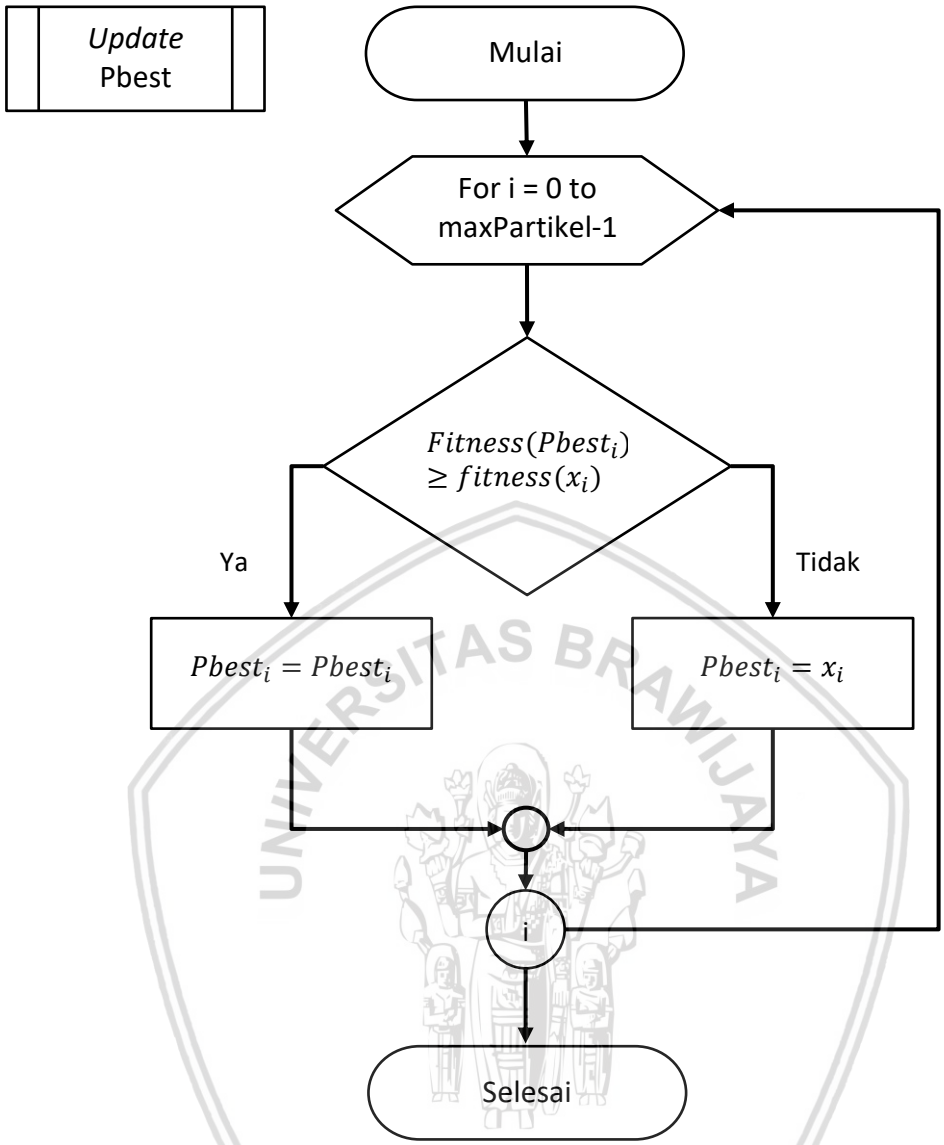


Gambar 4.5 Proses *Update* Partikel

1. Lakukan perulangan sebanyak partikel yang ada.
2. Lakukan perulangan sebanyak parameter yang ada pada partikel.
3. Lakukan *update* partikel dengan Persamaan 2.20.
4. Lakukan perhitungan *fitness* dengan fungsi pengujian akurasi.

4.2.1.4 Proses Update Pbest

Berikut merupakan bagian dari alur algoritme *Particle Swarm Optimization* pada sub proses *update Pbest*. Gambar 4.6 menjelaskan alur dari proses *update Pbest*.

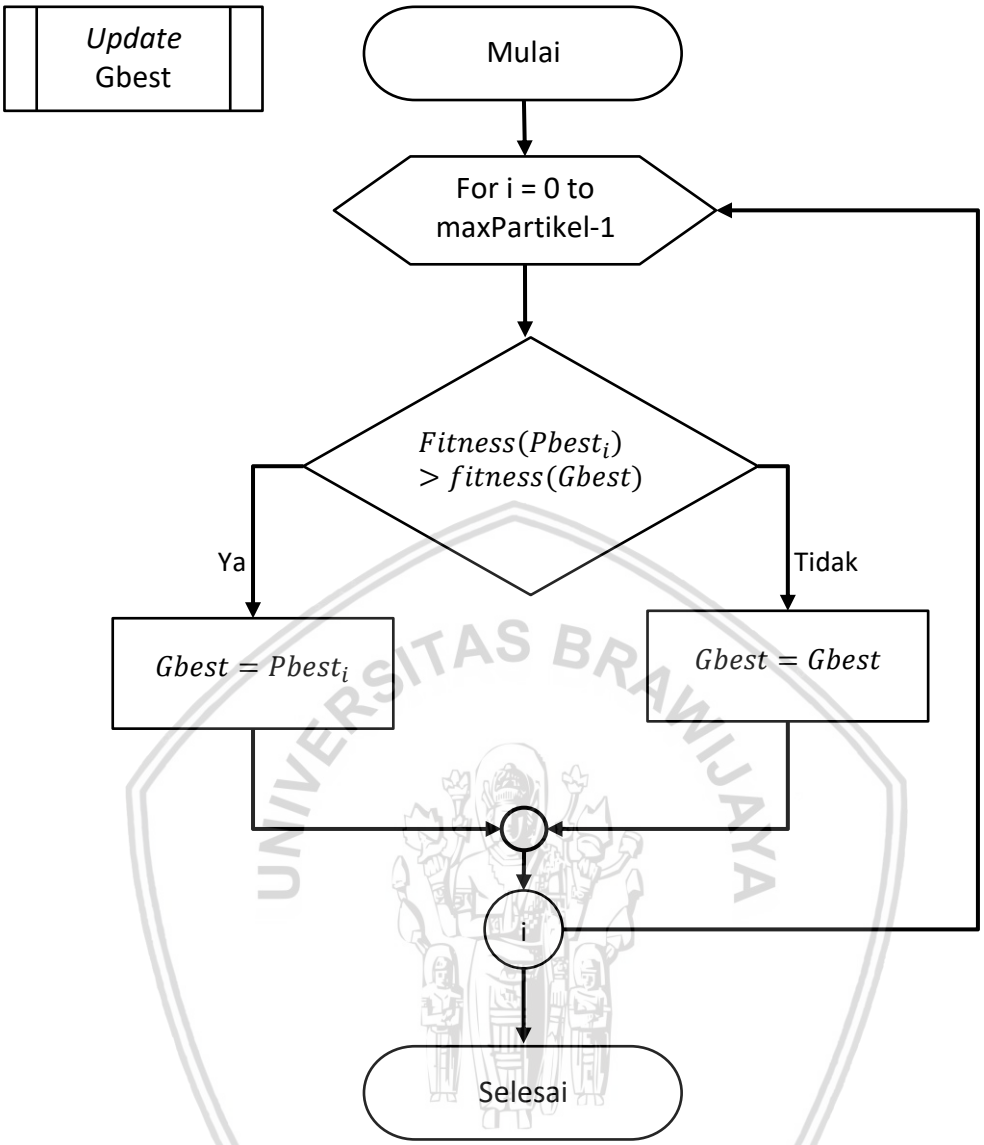


Gambar 4.6 Proses *Update Pbest*

1. Lakukan perulangan sebanyak partikel yang ada.
2. Lakukan *update Pbest* dengan Persamaan 2.21.

4.2.1.5 Proses Update Gbest

Berikut ini merupakan bagian dari alur algoritme *Particle Swarm Optimization* pada sub proses *update* Gbest. Gambar 4.7 menjelaskan alur dari proses *update Gbest*.

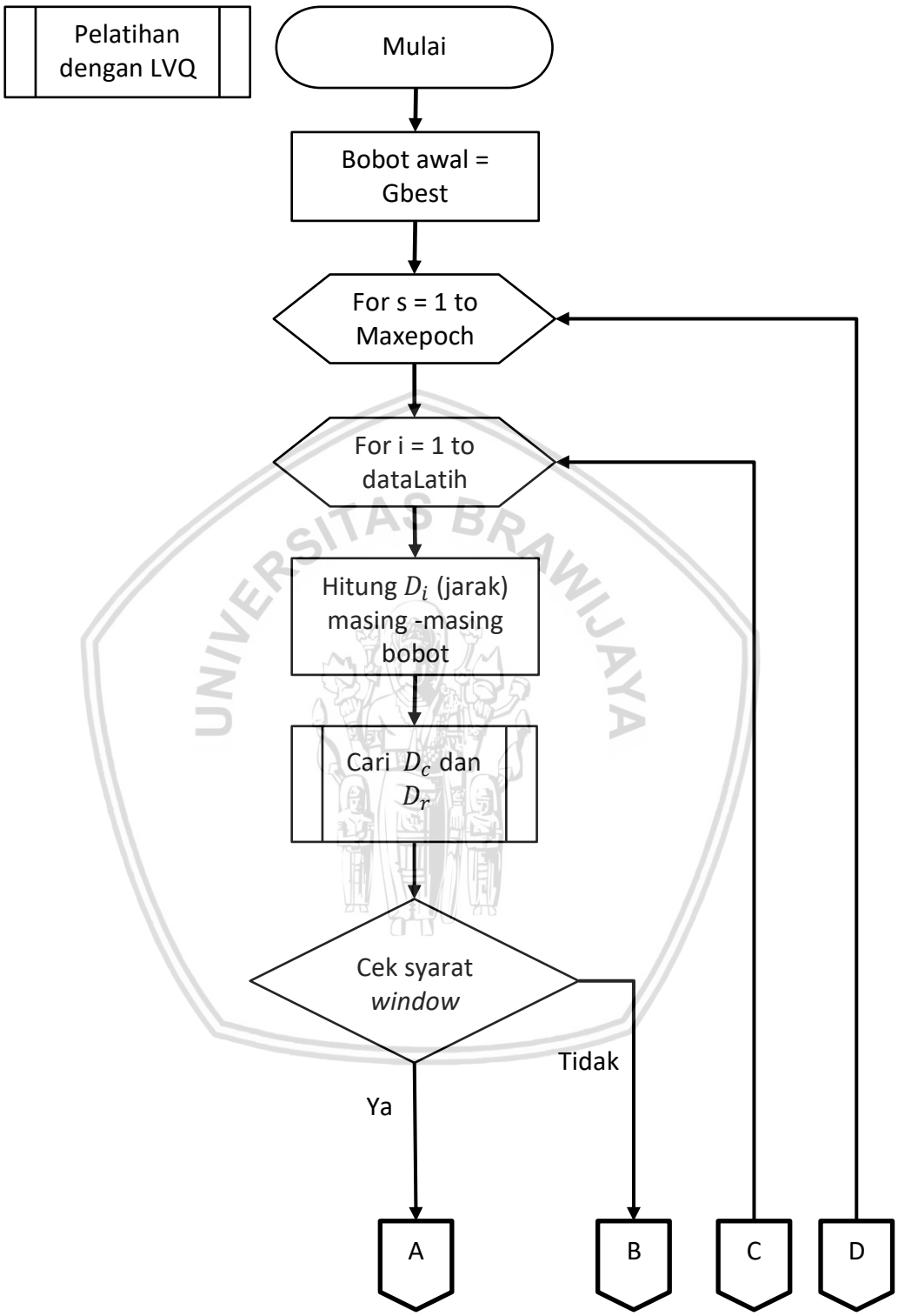


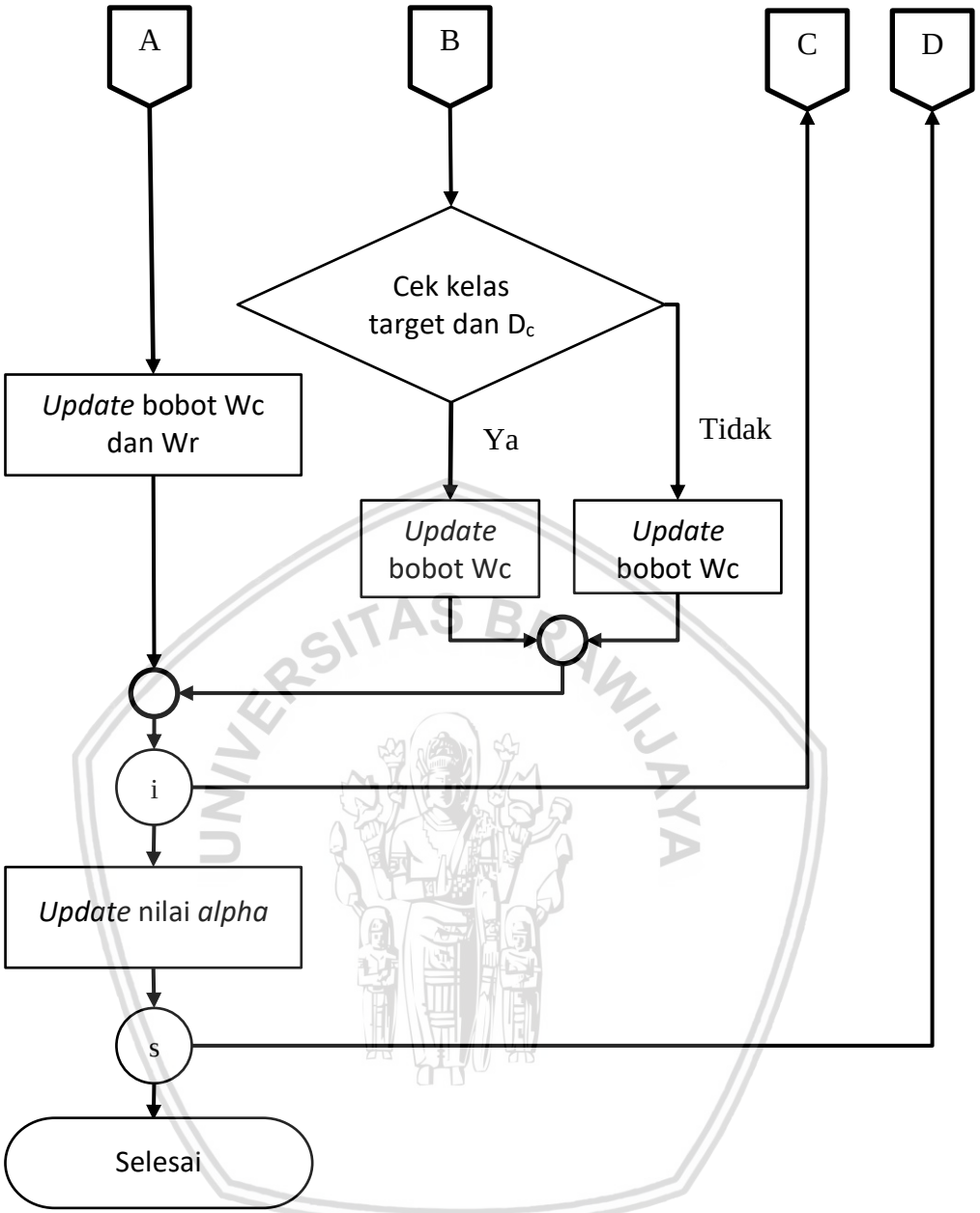
Gambar 4.7 Proses *Update Gbest*

1. Lakukan perulangan sebanyak partikel yang ada.
2. Lakukan *update Gbest* dengan Persamaan 2.22.

4.2.2 Proses *Learning Vector Quantization*

Pada proses algoritme *Learning Vector Quatization* ini akan dilakukan proses pelatihan dengan menggunakan bobot hasil perhitungan pada algoritme *Particle Swarm Optimization*, bobot yang diambil berasal dari perhitungan akhir dari *Gbest* terbaik yang didapat dari algoritme *Particle Swarm Optimization.* Bobot tersebut nantinya akan menjadi acuan dalam proses pelatihan dengan data latih yang sudah ada sehingga hasil dari proses ini berupa bobot terbaik. Untuk alur algoritme *Learning Vector Quantization* pada Gambar 4.8.


Pelatihan dengan LVQ
Mulai
Bobot awal = Gbest
For s = 1 to Maxepoch
For i = 1 to dataLatih
Hitung D_i (jarak) masing -masing bobot
Cari D_c dan D_r
Cek syarat *window*
Tidak
Ya
A
B
C
D



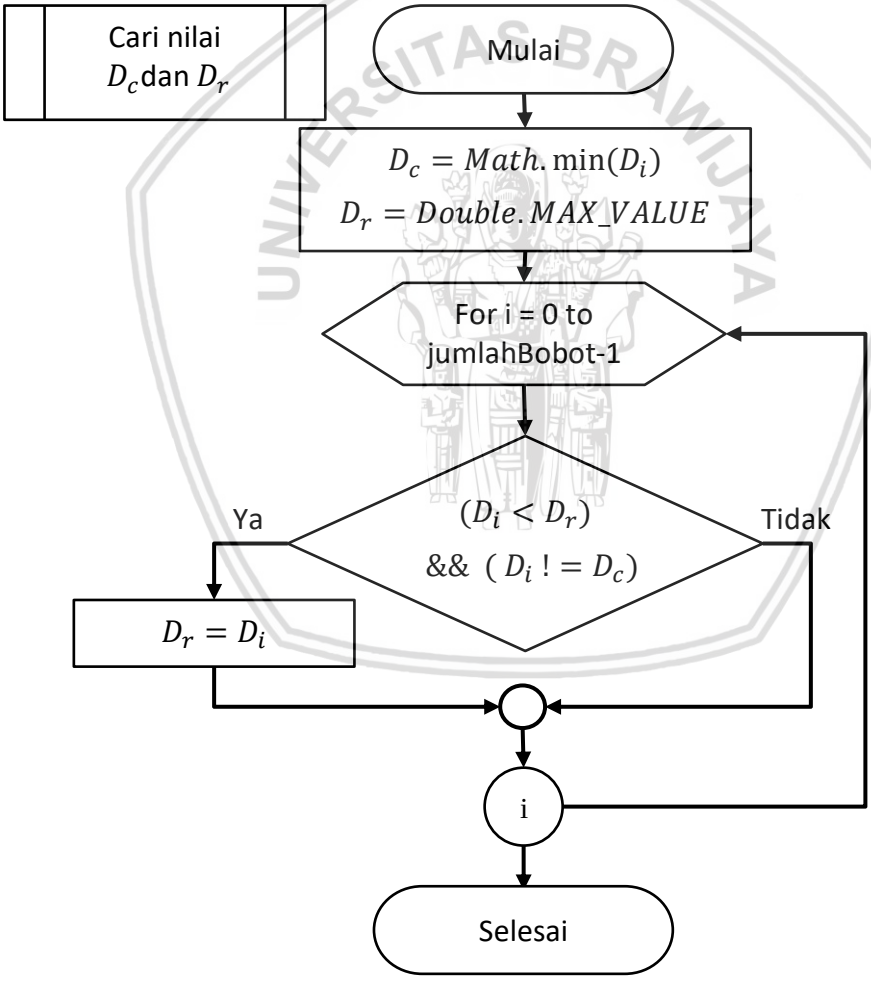
Gambar 4.8 Proses *Learning Vector Quantization*

1. Inisialisasi bobot awal dengan menggunakan nilai *Gbest.*
2. Lakukan perulangan sebanyak maxEpoch.
3. Lakukan perulangan sebanyak data latih yang digunakan.
4. Hitung jarak (D) dari setiap bobot terhadap data latih dengan Persamaan 2.3.
5. Cari jarak terkecil pertama (D_c) dan kedua (D_r) dari peritungan jarak yang telah dilakukan sebelumnya dengan menggunakan Persamaan 2.4 dan Persamaan 2.5.
6. Lakukan proses pengecekan nilai D_c dan D_r apakah sesuai dengan syarat window sesuai dengan Persamaan 2.6.

7. Apabila sesuai (benar) jarak pada window, lakukan *update* bobot W_c dan W_r sesuai Persamaan 2.7 dan Persamaan 2.8.
8. Apabila tidak sesuai (salah) maka lakukan pengecekan kelas pada D_c dan kelas data latih.
9. Bila kelas sama maka lakukan *update* bobot W_c dengan Persamaan 2.9.
10. Apabila tidak sesuai (salah) maka lakukan *update* bobot W_c dengan Persamaan 2.7.
11. *Update* nilai alpha sesuai Persamaan 2.10.

4.2.2.1 Proses Pencarian Nilai D_c dan D_r

Berikut ini merupakan bagian dari alur algoritme *Learning Vector Quantization* pada sub proses cari nilai D_c dan D_r. Gambar 4.9 menjelaskan alur dari proses cari nilai D_c dan D_r.



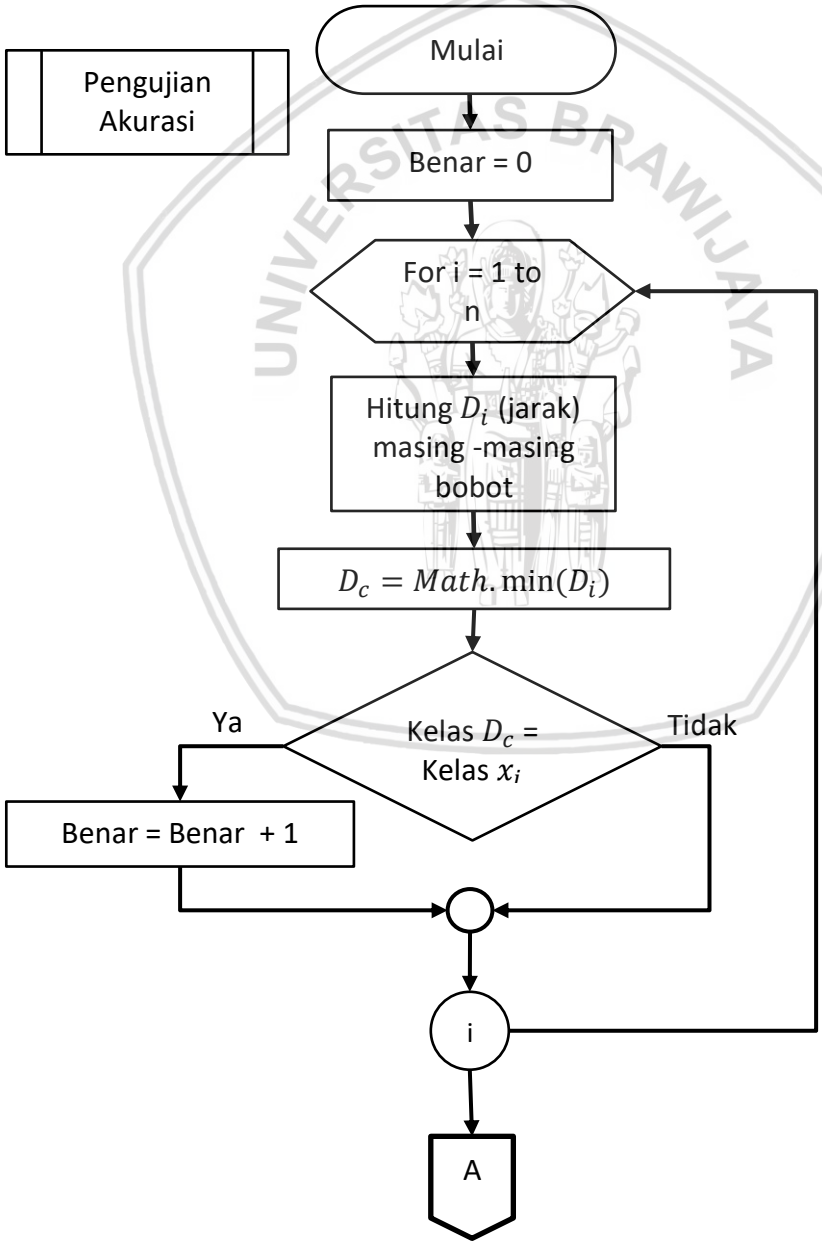
Gambar 4.9 Proses Pencarian Nilai D_c dan D_r

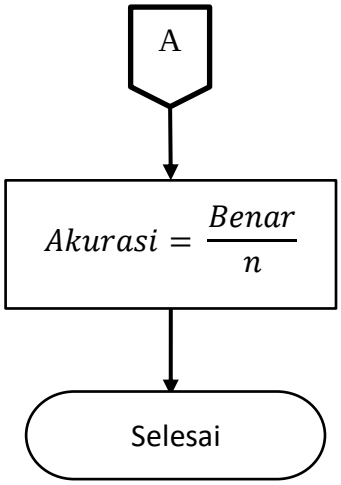
1. Hitung nilai D_c dengan menggunakan fungsi *Math.min*(D_i).
2. Beri nilai D_r dengan nilai maksimum *Double.*

3. Lakukan perulangan sebanyak jumlah bobot yang didefinisikan.
4. Apabila D_i kurang dari D_r dan D_i tidak sama dengan D_c maka nilai D_r yang baru adalah D_i.

4.2.3 Pengujian Akurasi

Proses pengujian akurasi ini digunakan untuk melihat akurasi dari bobot yang akan dihitung. Pengujian akurasi ini digunakan untuk mengetahui akurasi pada perhitungan nilai *fitness* pada algoritme *Particle Swarm Optimization* dan juga digunakan untuk mengetahui akurasi dari proses akhir pelatihan dengan algoritme *Learning Vector Quantization.* Alur pengujian dapat dilihat pada Gambar 4.10.





Gambar 4.10 Pengujian Akurasi

1. Berikan nilai awal variabel benar dengan nilai 0.
2. Lakukan perulangan sebanyak *n* (data latih) yang ada.
3. Hitung jarak masing-masing bobot dengan setiap data latih dengan menggunakan Persamaan 2.23.
4. Cari jarak terpendek dengan Persamaan 2.4 dan bandingkan kelas pada bobot dari jarak terpendek dengan kelas data latih. Apabila sesuai maka tambah nilai benar dengan satu.
5. Hitung akurasi dengan menggunakan Persamaan 2.24.

4.3 Perhitungan Manual

Perhitungan manual ini bertujuan untuk memberikan contoh perhitungan secara manual sehingga dapat memberikan gambaran bagaimana proses perhitungan yang nantinya akan dilakukan oleh kode program yang akan dibuat. Pada proses perhitungan manualisasi ini nilai parameter yang digunakan pada perhitungan manual *Particle Swarm Optimization* dan *Learning Vector Quantization* sebagai berikut:

Parameter pada *Particle Swarm Optimization:*

Iterasi	: 1
Popsize partikel	: 5
Bobot inersi	: 0,5
c1 dan c2	: 1
r1 dan r2	: 0,5

Parameter pada *Learning Vector Quantization:*

Iterasi : 2

Epsilon : 0,35

Alpha : 0,01

4.3.1 Perhitungan Manual Proses *Particle Swarm Optimization*

4.3.1.1 Tahap Inisialisasi

Pada tahap perhitungan *Particle Swarm Optimization* (PSO) dilakukan proses penentuan batas minimum dan juga maksimum dari partikel dan batas minimum dan juga maksimum dari kecepatan sehingga nilai partikel dan kecepatan tidak melewati batas yang ada.

Untuk batas minimum dan juga maksimum pada partikel didapatkan dari nilai minimum dan juga maksimum dari data latih yang digunakan, sehingga didapatkan nilai pada Tabel 4.2.

Tabel 4.2 Batas Minimum dan Maksimum Partikel

Batas Minimum dan Maksimum Partikel				
	Sepal Length	Sepal Width	Petal Length	Petal Width
Min	4,9	2,7	1,4	0,2
Max	7	3,5	6	2,5

Untuk batas minimum dan maskimum kecepatan sendiri didapatkan dengan menggunakan Persamaan 2.13 dan Persamaan 2.14. Pada Tabel 4.3 merupakan nilai minimun dan maksimum dari partikel dengan parameter *sepal length* yang akan digunakan sebagai contoh perhitungan berikut.

Tabel 4.3 Batas Minimum dan Maksimum Partikel pada Parameter *Sepal Length*

	Sepal Length
Min	4,9
Max	7

$$V_{Sepal\ length,max} = \frac{1}{2}\left(T_{j,max} - T_{j,min}\right)$$

$$= \frac{1}{2}(7 - 4{,}9)$$

$$= 1{,}05$$

$$V_{Sepal\ length,min} = -\frac{1}{2}(T_{j,max} - T_{j,min})$$
$$= -\frac{1}{2}(7 - 4{,}9)$$
$$= -1{,}05$$

Dari perhitungan tersebut didapatkan nilai batas kecepatan minimum dan juga maksimum pada Tabel 4.4:

Tabel 4.4 Batas Minimum dan Maksimum Kecepatan pada Parameter *Sepal Length*

	Sepal Length
Min	1,05
Max	-1,05

Dengan menghitung batas minimum dan juga maksimum pada seluruh parameter dengan menggunakan Persamaan 2.13 dan Persamaan 2.14 maka didapatkan nilai batas minimum dan juga maksimum pada Tabel 4.5.

Tabel 4.5 Batas Minimum dan Maksimum Kecepatan

Batas Minimum dan Maksimum *Kecepatan*				
	Sepal Length	Sepal Width	Petal Length	Petal Width
Min	1,05	0,4	2,3	1,15
Max	-1,05	-0,4	-2,3	-1,15

Selanjutnya dilakukan proses inisialisasi kecepatan awal, populasi awal partikel, nilai pbest dan gbest. Pada inisialisasi kecepatan ini setiap nilai kecepatan awal memiliki 3 nilai vektor kecepatan yang mewakili jumlah kelas pada data latih yang digunakan. Dengan menggunakan Persamaan 2.15 maka didapatkan nilai pada Tabel 4.6.

Tabel 4.6 Inisialisasi Kecepatan Awal

Inisialisasi Kecepatan Awal				
	Sepal length	Sepal Width	Petal Length	Petal Width
V1(0)	0	0	0	0
	0	0	0	0
	0	0	0	0
V2(0)	0	0	0	0
	0	0	0	0

Tabel 4.7 Inisialisasi Kecepatan Awal (lanjutan)

V2(0)	0	0	0	0
V3(0)	0	0	0	0
	0	0	0	0
	0	0	0	0
V4(0)	0	0	0	0
	0	0	0	0
	0	0	0	0
V5(0)	0	0	0	0
	0	0	0	0
	0	0	0	0

Selanjutnya adalah proses inisialisasi nilai partikel awal dengan menggunakan Persamaan 2.16. Pada proses inisialisasi ini setiap partikel diwakili dengan 3 nilai bobot yang mewakili jumlah kelas yang ada pada data latih. Contoh perhitungan nilai partikel awal w1 pada bobot *vector* pertama sebagai berikut:

Untuk particle w1 bobot pertama,

$$w1_{1,sepal\ length}(0) = T_{sepal,min} + rand[0,1]\ (T_{sepal,max} - T_{sepal,\ min})$$
$$= 4{,}9 + 0{,}510691684(7 - 4{,}9)$$
$$= 4{,}9 + 0{,}510691684(2{,}1)$$
$$= 4{,}9 + 1{,}072452536$$
$$= 5{,}972452536$$

$$w1_{1,sepal\ width}(0) = T_{sepal,min} + rand[0,1]\ (T_{sepal,max} - T_{sepal,\ min})$$
$$= 2{,}7 + 0{,}407087580(3{,}5 - 2{,}7)$$
$$= 2{,}7 + 0{,}407087580(0{,}8)$$
$$= 2{,}7 + 0{,}325670064$$
$$= 3{,}025670064$$

$$w1_{1,petal\ length}(0) = T_{petal,min} + rand[0,1]\ (T_{petal,max} - T_{petal,\ min})$$
$$= 1{,}4 + 0{,}099632991(6 - 1{,}4)$$
$$= 1{,}4 + 0{,}099632991(4{,}6)$$
$$= 1{,}4 + 0{,}458311757$$
$$= 1{,}858311757$$

$$w1_{1,petal\ width}(0) = T_{petal,min} + rand[0,1]\ (T_{petal,max} - T_{petal,\ min})$$
$$= 0{,}2 + 0{,}398931680(2{,}5 - 0{,}2)$$

$$= 0{,}2 + 0{,}398931680(2{,}3)$$
$$= 0{,}2 + 0{,}917542864$$
$$= 1{,}117542864$$

Setelah menghitung bobot pada partikel w1, didapatkan nilai pada Tabel 4.7.

Tabel 4.8 Inisialisasi Partikel x1 Bobot Pertama

	Sepal length	Sepal Width	Petal Length	Petal Width
w1(0)	5,972452536	3,025670064	1,858311757	1,117542864
	-	-	-	-
	-	-	-	-

Selanjutnya lakukan perhitungan pada bobot lainnya dan juga pada partikel lain dengan menggunakan Persamaan 2.16 sehingga didapatkan nilai pada Tabel 4.8.

Tabel 4.9 Inisialisasi Partikel Awal

Inisialisasi Partikel Awal				
	Sepal length	Sepal Width	Petal Length	Petal Width
w1(0)	5,972452536	3,025670064	1,858311757	1,117542864
	5,729496669	2,89103111	1,567670545	1,986342912
	6,412990962	2,975859148	3,986027161	1,826988935
w2(0)	5,615036893	3,021980232	4,352983791	2,476899142
	4,98111127	2,983736417	3,624553335	1,436736644
	6,551820056	2,905438122	2,39959986	2,471302217
w3(0)	6,80705237	3,481021451	3,596896826	0,821258802
	5,339715272	2,933901056	4,762568372	1,46489315
	5,144789539	3,292308896	3,253712419	1,363644191
w4(0)	5,679697827	3,40913282	3,367018286	1,022697808
	5,190604176	2,906504121	2,653808277	0,328554286
	4,91794891	3,373933196	5,713929782	1,138860663
w5(0)	6,217526762	3,24128622	4,022443861	0,743425816
	5,502200265	3,309020985	3,747502729	1,899095624
	5,896013021	3,205245184	3,300503958	1,065020966

Setelah seluruh bobot partikel awal telah didapatka selanjutnya lakukan perhitungan nilai *fitness* dengan menggunakan Persamaan 2.23 dan Persamaan 2.24. Perhitungan fitness dilakukan dengan menghitung kedekatan data latih pada Tabel 4.1 dengan seluruh partikel pada Tabel 4.8. Berikut merupakan contoh perhitungan nilai *fitness* dengan menggunakan bobot partikel w1 dan juga data latih pertama.

Tabel 4.10 Bobot Partikel w1

	Sepal length	Sepal Width	Petal Length	Petal Width	*Fitness*
w1(0)	5,972452536	3,025670064	1,858311757	1,117542864	-
	5,729496669	2,89103111	1,567670545	1,986342912	
	6,412990962	2,975859148	3,986027161	1,826988935	

Tabel 4.11 Data latih pertama

No	Sepal Length	Sepal Width	Petal Length	Petal Width	Kelas	
1.	5,1	3,5	1,4	0,2	Iris-setosa	1

Perhitungan untuk partikel 1 bobot *vector* ke-1 dengan data latih pertama:

$$\boldsymbol{D_{1,1}} = \sqrt{\sum_{j=1}^{n}\left(\boldsymbol{X_j} - \boldsymbol{W_{1j}}\right)^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 5{,}972452536)^2 + (3{,}5 - 3{,}025670064)^2 \\ +(1{,}4 - 1{,}858311757)^2 + (0{,}2 - 1{,}117542864)^2]\end{matrix}}$$

$$= 1{,}427619309$$

Perhitungan untuk partikel 1 bobot *vector* ke-2:

$$\boldsymbol{D_{1,2}} = \sqrt{\sum_{j=1}^{n}\left(\boldsymbol{X_j} - \boldsymbol{W_{2j}}\right)^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 5{,}729496669)^2 + (3{,}5 - 2{,}89103111)^2 \\ +(1{,}4 - 1{,}567670545)^2 + (0{,}2 - 1{,}986342912)^2]\end{matrix}}$$

$$= 1{,}996557932$$

Perhitungan untuk partikel 1 bobot *vector* ke-3:

$$D_{1,3} = \sqrt{\sum_{j=1}^{n}(X_j - W_{3j})^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 6{,}412990962)^2 + (3{,}5 - 2{,}975859148)^2 \\ +(1{,}4 - 3{,}986027161)^2 + (0{,}2 - 1{,}826988935)^2]\end{matrix}}$$

$$= 3{,}366496454$$

Selanjutnya cari nilai terkecil dari perhitungan jarak yang telah dilakukan pada partikel 1 dengan Persamaan 2.4. Untuk data jarak yang telah dihitung berada pada Tabel 4.11.

Tabel 4.12 Jarak bobot pada partikel w1 dengan data latih pertama

Bobot	jarak
Bobot 1	1,427619309
Bobot 2	1,996557932
Bobot 3	3,366496454

Dari Tabel 4.11 didapat bahwa bobot terdekat dengan data latih pertama adalah bobot ke-1, sehingga partikel w1 merupakan kelas ke-1. Kemudian ulangi langkah perhitungan jarak bobot dengan data latih yang lain sehingga didapatkan data pada Tabel 4.12.

Tabel 4.13 Jarak partikel w1 terhadap seluruh data latih

Data latih	Kelas berdasarkan Jarak Bobot Terdekat	Kelas Data latih	Keseuaian
1	1	1	1
2	1	1	1
3	3	2	0
4	3	2	0
5	3	3	1
6	3	3	1
Total			4

Setelah itu lakukan perhitungan dengan Persamaan 2.24 sehingga didapatkan nilai *fitness* dari partikel *w1*.

$$\boldsymbol{Akurasi} = \frac{\boldsymbol{c}}{\boldsymbol{n}}$$
$$= \frac{4}{6}$$
$$= 0{,}666666667$$

Nilai akurasi yang didapat digunakan sebagai nilai *fitness* pada partikel. Selanjutnya hitung seluruh nilai *fitness* pada partikel lain, dan didapatkan nilai pada Tabel 4.13.

Tabel 4.14 Inisialisasi Partikel Awal

Inisiasliasi Particle Awal					
	Sepal length	Sepal width	Petal length	Petal width	*Fitness*
w1(0)	5,972452536	3,025670064	1,858311757	1,117542864	0,666667
	5,729496669	2,89103111	1,567670545	1,986342912	
	6,412990962	2,975859148	3,986027161	1,826988935	
w2(0)	5,615036893	3,021980232	4,352983791	2,476899142	0
	4,98111127	2,983736417	3,624553335	1,436736644	
	6,551820056	2,905438122	2,39959986	2,471302217	
w3(0)	6,80705237	3,481021451	3,596896826	0,821258802	0,166667
	5,339715272	2,933901056	4,762568372	1,46489315	
	5,144789539	3,292308896	3,253712419	1,363644191	
w4(0)	5,679697827	3,40913282	3,367018286	1,022697808	0,333333
	5,190604176	2,906504121	2,653808277	0,328554286	
	4,91794891	3,373933196	5,713929782	1,138860663	
w5(0)	6,217526762	3,24128622	4,022443861	0,743425816	0
	5,502200265	3,309020985	3,747502729	1,899095624	
	5,896013021	3,205245184	3,300503958	1,065020966	

Setelah melakukan perhitungan nilai partikel awal, selanjutnya dilakukan inisialisasi terhadap nilai pbest. Inisialisasi Pbest dilakukan dengan menggunakan Persamaan 2.17. Pada persamaan tersebut dijelaskan bahwa inisialisasi Pbest awal memiliki nilai yang sama dengan nilai partikel awal, sehingga didapatkan nilai Pbest pada Tabel 4.14

Tabel 4.15 Inisialisasi Pbest

Inisialisasi Pbest					
	Sepal length	Sepal width	Petal length	Petal width	*Fitness*
Pbest1(0)	5,972452536	3,025670064	1,858311757	1,117542864	0,666667
	5,729496669	2,89103111	1,567670545	1,986342912	
	6,412990962	2,975859148	3,986027161	1,826988935	
Pbest 2(0)	5,615036893	3,021980232	4,352983791	2,476899142	0
	4,98111127	2,983736417	3,624553335	1,436736644	
	6,551820056	2,905438122	2,39959986	2,471302217	
Pbest 3(0)	6,80705237	3,481021451	3,596896826	0,821258802	0,166667
	5,339715272	2,933901056	4,762568372	1,46489315	
	5,144789539	3,292308896	3,253712419	1,363644191	
Pbest 4(0)	5,679697827	3,40913282	3,367018286	1,022697808	0,333333
	5,190604176	2,906504121	2,653808277	0,328554286	
	4,91794891	3,373933196	5,713929782	1,138860663	
Pbest 5(0)	6,217526762	3,24128622	4,022443861	0,743425816	0
	5,502200265	3,309020985	3,747502729	1,899095624	
	5,896013021	3,205245184	3,300503958	1,065020966	

Tahap selanjutnya tentukan nilai Gbest dengan menggunakan Persamaan 2.18. Pada persamaan tersebut nilai Gbest didapatkan dengan mencari nilai terbaik dari fitness yang ada dan didapat Gbest terbaik pada Tabel 4.15.

Tabel 4.16 Inisialisasi Gbest

Inisialisasi Gbest					
	Sepal length	Sepal width	Petal length	Petal width	*Fitness*
Gbest(0)	5,972452536	3,025670064	1,858311757	1,117542864	0,666667
	5,729496669	2,89103111	1,567670545	1,986342912	
	6,412990962	2,975859148	3,986027161	1,826988935	

4.3.1.2 Tahap Update

Setelah melakukan proses inisialisasi data, dilanjutkan pada tahap *update* data. *Update* data yang dilakukan adalah *update* kecepatan, partikel, Pbest dan Gbest. Untuk proses *update* kecepatan gunakan Persamaan 2.19. *Update* kecepatan sendiri digunakan untuk merubah nilai partikel sehingga akan didapatkan nilai partikel terbaik. Sebagai contoh proses *update* kecepatan, digunakan nilai kecepatan v1 untuk *vector* pertama pada Tabel 4.6, nilai Pbest1

pada Tabel 4.14, nilai Gbest pada Tabel 4.15 dan juga dengan nilai bobot inersi (w) adalah 0,5, nilai *c1* dan *c2* adalah 1, dan juga nilai *r1* dan *r2* adalah 0,5.

$$v^1_{1,Sepal\ length} = w.v^0_{1,Sepal} + c_1.r_1(Pbest^0_{1,sepal} - x^0_{1,sepal}) + c_2.r_2(Gbest^0_{sepal} - x^0_{1,sepal})$$
$$= 0{,}5 * 0 + 1 * 0{,}5(5{,}972452536 - 5{,}972452536) + 1 * 0{,}5(5{,}972452536 - 5{,}972452536)$$
$$= 0$$

$$v^1_{1,Sepal\ width} = w.v^0_{1,Sepal} + c_1.r_1(Pbest^0_{1,sepal} - x^0_{1,sepal}) + c_2.r_2(Gbest^0_{sepal} - x^0_{1,sepal})$$
$$= 0{,}5 * 0 + 1 * 0{,}5(3{,}025670064 - 3{,}025670064) + 1 * 0{,}5(3{,}025670064 - 3{,}025670064)$$
$$= 0$$

$$v^1_{1,Petal\ length} = w.v^0_{1,Petal} + c_1.r_1(Pbest^0_{1,petal} - x^0_{1,petal}) + c_2.r_2(Gbest^0_{petal} - x^0_{1,petal})$$
$$= 0{,}5 * 0 + 1 * 0{,}5(1{,}858311757 - 1{,}858311757) + 1 * 0{,}5(1{,}858311757 - 1{,}858311757)$$
$$= 0$$

$$v^1_{1,Petal\ width} = w.v^0_{1,Petal} + c_1.r_1(Pbest^0_{1,petal} - x^0_{1,petal}) + c_2.r_2(Gbest^0_{petal} - x^0_{1,petal})$$
$$= 0{,}5 * 0 + 1 * 0{,}5(1{,}117542864 - 1{,}117542864) + 1 * 0{,}5(1{,}117542864 - 1{,}117542864)$$
$$= 0$$

Dengan perhitungan diatas didapatkan nilai V1 untuk *vector* pertama pada Tabel 4.16.

Tabel 4.17 *Update* Kecepatan V1 pada *vector* pertama

Update Kecepatan				
	Sepal length	Sepal Width	Petal Length	Petal Width
V1(1)	0	0	0	0
	-	-	-	-
	-	-	-	-

Lakukan perhitungan *update* kecepatan dengan Persamaan 2.19 pada seluruh data kecepatan pada Tabel 4.6 sehingga didapatkan nilai pada Tabel 4.17.

Tabel 4.18 *Update* Kecepatan

Update Kecepatan				
	Sepal length	Sepal Width	Petal Length	Petal Width
V1(1)	0	0	0	0
	0	0	0	0
	0	0	0	0
V2(1)	0,178707822	0,001844916	-1,247336017	-0,679678139
	0,374192699	-0,046352654	-1,028441395	0,274803134
	-0,069414547	0,035210513	0,793213651	-0,322156641
V3(1)	-0,417299917	-0,227675693	-0,869292535	0,148142031
	0,194890699	-0,021434973	-1,597448913	0,260724881
	0,634100711	-0,158224874	0,366157371	0,231672372
V4(1)	0,146377354	-0,191731378	-0,754353265	0,047422528
	0,269446246	-0,007736506	-0,543068866	0,828894313
	0,747521026	-0,199037024	-0,86395131	0,344064136
V5(1)	-0,122537113	-0,107808078	-1,082066052	0,187058524
	0,113648202	-0,208994938	-1,089916092	0,043623644
	0,25848897	-0,114693018	0,342761602	0,380983985

Langkah selanjutnya lakukan *update* terhadap partikel pada Tabel 4.13. proses *update* partikel menggunakan Persamaan 2.20. Untuk proses *update* sendiri akan digunakan nilai pada partikel w1 pada Tabel 4.13 dengan nilai kecepatan V1 yang telah di *update* pada Tabel 4.16 untuk *vector* pertama.

$$x^1_{1,sepal\ length} = x^0_{1,sepal} + v^1_{1,sepal}$$

$$= 5{,}972452536 + 0$$

$$= 5{,}972452536$$

$$x^1_{1,sepal\ width} = x^0_{1,sepal} + v^1_{1,sepal}$$

$$= 3{,}025670064 + 0$$

$$= 3{,}025670064$$

$$x^1_{1,petal\ length} = x^0_{1,petal} + v^1_{1,petal}$$

$$= 1{,}858311757 + 0$$

$$= 1{,}858311757$$

$$x^1_{1,petal\ width} = x^0_{1,petal} + v^1_{1,petal}$$

$$= 1{,}117542864 + 0$$

$$= 1{,}117542864$$

Ulangi perhitungan diatas pada seluruh vector partikel w1 sehingga didapatkan nilai partikel w1 *update*. Berikut merupakan perbandingan nilai pada partikel w1 sebelum dilakukan *update* dengan partikel w1 hasil *update* pada Tabel 4.18.

Tabel 4.19 Perbandingan Nilai Partikel Sebelum dan Sesudah *Update*

	Sepal length	sepal width	petal length	petal width
w1(0)	5,972452536	3,025670064	1,858311757	1,117542864
	5,729496669	2,89103111	1,567670545	1,986342912
	6,412990962	2,975859148	3,986027161	1,826988935
w1(1)	5,972452536	3,025670064	1,858311757	1,117542864
	5,729496669	2,89103111	1,567670545	1,986342912
	6,412990962	2,975859148	3,986027161	1,826988935

Dengan menggunakan langkah yang sama dan dilakukan pada seluruh nilai partikel maka didapatkan nilai partikel pada Tabel 4.19 dan tak lupa hitung kembali nilai fitness dengan cara yang sama pada tahap inisialisasi dengan menggunakan Persamaan 2.23 dan Persamaan 2.24.

Tabel 4.20 Partikel *Update*

Partikel *Update*					
	Sepal length	sepal width	petal length	petal width	*Fitness*
w1(1)	5,972452536	3,025670064	1,858311757	1,117542864	0,666667
	5,729496669	2,89103111	1,567670545	1,986342912	
	6,412990962	2,975859148	3,986027161	1,826988935	
w2(1)	5,793744714	3,023825148	3,105647774	1,797221003	0,166667
	5,35530397	2,937383763	2,59611194	1,711539778	

Tabel 4.21 Partikel *Update* (lanjutan)

w2(1)	6,482405509	2,940648635	3,192813511	2,149145576	
w3(1)	6,389752453	3,253345757	2,727604291	0,969400833	0,666667
	5,53460597	2,912466083	3,165119459	1,725618031	
	5,77889025	3,134084022	3,61986979	1,595316563	
w4(1)	5,826075182	3,217401442	2,612665021	1,070120336	0,333333
	5,460050422	2,898767616	2,110739411	1,157448599	
	5,665469936	3,174896172	4,849978472	1,482924799	
w5(1)	6,094989649	3,133478142	2,940377809	0,93048434	0,666667
	5,615848467	3,100026047	2,657586637	1,942719268	
	6,154501991	3,090552166	3,64326556	1,446004951	

Setelah melakukan *update* terhadap nilai partikel, selanjutnya dilakukan *update* pada nilai pbest dan best. Untuk melakukan *update* nilai pada pbest digunakan persamaan 2.21 dengan membandikan nilai *fitness* Pbest sebelumnya dengan nilai partikel yang telah menjalani proses *update*. Untuk proses *update* Pbest berada pada Tabel 4.20.

Tabel 4.22 Perbandingan nilai *fitness*

Fitness Pbest(0)		*Fitness* Partikel(1)		Pbest(1)
Pbest1(0)	0,666666667	w1(1)	0,666666667	Pbest1(0)
Pbest2(0)	0	w2(1)	0,166666667	w2(1)
Pbest3(0)	0,166666667	w3(1)	0,666666667	w3(1)
Pbest4(0)	0,333333333	w4(1)	0,333333	Pbest4(0)
Pbest5(0)	0	w5(1)	0,666666667	w5(1)

Dengan melakukan perbandingan tersebut didapatkan nilai Pbest(1) pada Tabel 4.21.

Tabel 4.23 Pbest *Update*

Pbest *Update*					
	Sepal length	sepal width	petal length	petal width	*Fitness*
Pbest1(1)	5,972452536	3,025670064	1,858311757	1,117542864	0,666666667
	5,729496669	2,89103111	1,567670545	1,986342912	
	6,412990962	2,975859148	3,986027161	1,826988935	

Tabel 4.24 Pbest *Update* (lanjutan)

Pbest2(1)	5,793744714	3,023825148	3,105647774	1,797221003	0,166666667
	5,35530397	2,937383763	2,59611194	1,711539778	
	6,482405509	2,940648635	3,192813511	2,149145576	
Pbest3(1)	6,389752453	3,253345757	2,727604291	0,969400833	0,666666667
	5,53460597	2,912466083	3,165119459	1,725618031	
	5,77889025	3,134084022	3,61986979	1,595316563	
Pbest4(1)	5,826075182	3,217401442	2,612665021	1,070120336	0,333333333
	5,460050422	2,898767616	2,110739411	1,157448599	
	5,665469936	3,174896172	4,849978472	1,482924799	
Pbest5(1)	6,094989649	3,133478142	2,940377809	0,93048434	0,666666667
	5,615848467	3,100026047	2,657586637	1,942719268	
	6,154501991	3,090552166	3,64326556	1,446004951	

Tahap selanjutnya tentukan nilai Gbest dengan menggunakan Persamaan 2.18. Pada persamaan tersebut nilai Gbest didapatkan dengan mencari nilai terbaik dari fitness yang ada dan didapat Gbest terbaik pada Tabel 4.22.

Tabel 4.25 Gbest *Update*

	Gbest				
	Sepal length	sepal width	petal length	petal width	*Fitness*
Gbest(1)	5,972452536	3,025670064	1,858311757	1,117542864	0,6666667
	5,729496669	2,89103111	1,567670545	1,986342912	
	6,412990962	2,975859148	3,986027161	1,826988935	

4.3.2 Perhitungan Manual Proses *Learning Vector Quantization*

4.3.2.1 Tahap Inisialisasai

Pada tahap perhitungan *Learning Vector Quantization* (LVQ) dilakukan proses inisialisasi bobot yang mana bobot tersebut berasal dari hasil Gbest pada proses *Particle Swarm Optimization* (PSO) pada Tabel 4.22, sehingga didapatkan nilai awal *Learning Vector Quantization* pada Tabel 4.23.

Tabel 4.26 Bobot Awal LVQ

	Sepal length	sepal width	petal length	petal width
W1	5,972452536	3,025670064	1,858311757	1,117542864
W2	5,729496669	2,89103111	1,567670545	1,986342912
W3	6,412990962	2,975859148	3,986027161	1,826988935

4.3.2.2 Tahap Pelatihan

Setelah bobot awal telah didapat, selanjutnya adalah proses pelatihan bobot berdasarkan data yang ada pada Tabel 4.1, proses pelatihan tersebut bertujuan untuk mendapatkan bobot terbaik dari data yang ada. Misalkan bobot pada Tabel 4.23 akan dilatih dengan data pertama pada Tabel 4.1 dengan menggunakan Persamaan 2.3.

$$D_1 = \sqrt{\sum_{j=1}^{n}(X_j - W_{1j})^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 5{,}055454098)^2 + (3{,}5 - 2{,}979169924)^2 \\ +(1{,}4 - 3{,}400511434)^2 + (0{,}2 - 0{,}839858329)^2]\end{matrix}}$$

$$= 1{,}427619309$$

$$D_2 = \sqrt{\sum_{j=1}^{n}(X_j - W_{2j})^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 6{,}507326482)^2 + (3{,}5 - 2{,}918632853)^2 \\ +(1{,}4 - 4{,}205917049)^2 + (0{,}2 - 1{,}138435155)^2]\end{matrix}}$$

$$= 1{,}996557932$$

$$D_3 = \sqrt{\sum_{j=1}^{n}(X_j - W_{3j})^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 6{,}455989475)^2 + (3{,}5 - 3{,}267185466)^2 \\ +(1{,}4 - 4{,}507333961)^2 + (0{,}2 - 1{,}777985031)^2]\end{matrix}}$$

$$= 3{,}366496454$$

Dari perhitungan diatas didapatkan nilai jarak antara bobot dan data latih pertama pada Tabel 4.24.

Tabel 4.27 Jarak Antaran Bobot dan Data Latih Pertama

	jarak
W1	1,427619309
W2	1,996557932
W3	3,366496454

Setelah mendapatkan jarak dari setiap bobot, selanjutnya mencari jarak terkecil 1 dan terkecil 2 (D_c, D_r) dari jarak yang ada pada Tabel 4.24 dan dengan menggunakan Persamaan 2.4 dan Persamaan 2.5, didapatkan:

Tabel 4.28 Jarak D_c dan D_r

jarak		
D_c	Bobot 1	1,427619309
D_r	Bobot 2	1,996557932

Selanjutnya dilakukan pengecekan terhadap nilai $\boldsymbol{D_c}$ dan $\boldsymbol{D_r}$ yang telah dicari dengan persyaratan pada window pada Persamaan 2.6. Pada proses ini digunakan nilai epsilon (ε) = 0,35.

Syarat epsilon pertama:

$$\frac{D_c}{D_r} > 1 - \varepsilon$$

$$\frac{1{,}427619309}{1{,}996557932} > 1 - 0{,}35$$

$$0{,}715040263 > \mathbf{0{,}65}$$

Syarat epsilon kedua:

$$\frac{D_r}{D_c} < 1 + \varepsilon$$

$$\frac{1{,}996557932}{1{,}427619309} < 1 + 0{,}35$$

$$1{,}398522645 < \mathbf{1{,}35}$$

Selain melakukan pengecekan terhadap jarak data latih dan bobot dengan menggunakan *window*, dilakukan pengecekan kelas antara bobot pada jarak terpendek kedua dan data latih. Setiap bobot mewakili kelas yang digunakan seperti bobot *w1* mewakili kelas 1 (*Iris-setosa*), bobot *w2* mewakili kelas 2 (*Iris-versicolor*), dan bobot *w3* mewakili kelas 3 (*Iris-virginica*). Apabila memenuhi persyaratan yang ada maka bobot pada jarak terpendek pertama dan kedua akan di-*update* secara simultan dengan Persamaan 2.7 dan Persamaan 2.8. Namun bila tidak memenuhi maka dilakukan *update* pada jarak terpendek pertama.

Hasil dari pengecekan dengan persyaratan yang ada didapatkan bahwa jarak terpendek pertama dan jarak terpendek kedua dari data latih dan bobot tidak memenuhi persyaratan *epsilon* kedua maka *update* bobot dilakukan pada bobot dengan jarak terpendek pertama. Karena bobot dan data latih memiliki kelas yang sama maka bobot akan di-*update* dengan Persamaan 2.9. Pada proses perhitungan ini digunakan nilai *alpha* (α) = 0,01 dengan data latih pertama pada Tabel 4.1.

$$\mathbf{W_c(t+1) = W_c(t) + \alpha\,(t)[X_1(t) - W_c(t)]}$$

$$= \begin{matrix}5{,}972452536\\3{,}025670064\\1{,}858311757\\1{,}117542864\end{matrix} + \mathbf{0,01}\left[\begin{matrix}5{,}1\\3{,}5\\1{,}4\\0{,}2\end{matrix} - \begin{matrix}5{,}972452536\\3{,}025670064\\1{,}858311757\\1{,}117542864\end{matrix}\right]$$

$$= \begin{matrix}5{,}963728011\\3{,}030413363\\1{,}853728639\\1{,}108367436\end{matrix}$$

Maka bobot terbaru hasil pelatihan dengan data 1 adalah

Tabel 4.29 Bobot LVQ *Update*

Bobot LVQ				
	Sepal length	sepal width	petal length	petal width
W1	5,963728011	3,030413363	1,853728639	1,108367436
W2	5,729496669	2,89103111	1,567670545	1,986342912
W3	6,412990962	2,975859148	3,986027161	1,826988935

Lakukan pelatihan bobot dengan seluruh data latih yang ada pada Tabel 4.1 dengan menggunakan langkah-langkah yang sama dengan proses sebelumnya sehingga didapatkan nilai bobot pada iterasi pertama pada Tabel 4.27.

Tabel 4.30 Hasil Bobot Pelatihan LVQ iterasi pertama

Bobot LVQ				
	Sepal length	sepal width	petal length	petal width
W1	5,953090731	3,03010923	1,849191353	1,099283761
W2	5,729496669	2,89103111	1,567670545	1,986342912
W3	6,400058958	2,971893974	4,005000149	1,841813442

Tahap selanjutnya adalah *update* nilai alpha setiap iterasi selesai dengan Persamaan 2.10 sehingga akan didapatkan nilai alpha baru untuk proses pelatihan pada iterasi selanjutnya. Karena jumlah iterasi maksimum pada pelatihan ini adalah 2 kali iterasi maka didapat nilai alpha baru berikut:

$$\alpha(1) = \alpha(init) * [1 - ({}^{i}/_{m})]$$
$$= 0{,}01 * \left[1 - ({}^{1}/_{2})\right]$$
$$= 0{,}005$$

Lakukan proses pelatihan kembali pada bobot LVQ dengan nilai alpha yang telah diperbaharui. Pada Tabel 4.28 merupakan hasil akhir dari pelatihan dengan algoritme LVQ.

Tabel 4.31 Hasil Akhir Bobot Pelatihan LVQ

Bobot LVQ				
	Sepal length	sepal width	petal length	petal width
W1	5,943581151	3,03229639	1,844710669	1,090313406
W2	5,729496669	2,89103111	1,567670545	1,986342912
W3	6,393576529	2,969902879	4,014492423	1,849268839

4.3.3 Perhitungan Manual Proses Evaluasi

Pada tahap ini merupakan tahap terakhir dari algoritme ini, dimana akan dilakukan proses perhitungan akurasi dari bobot hasil proses algoritme *Particle Swarm Optimization-Learning Vector Quatization* (PSO-LVQ) dengan menggunakan Persamaan 2.23, Persamaan 2.24, dan Persamaan 2.25.

Pada Persamaan 2.23 dilakukan perhitungan jarak antara bobot hasil dari proses algoritme pada Tabel 4.28 dengan seluruh data latih pada Tabel 4.1. Selanjutnya periksa kelas hasil pencarian jarak terpendek dengan kelas data latih, apakah sesuai dengan kelas data latih atau tidak. Berikut merupakan contoh perhitungan dengan menggunakan bobot pada Tabel 4.28 dengan data latih pertama pada Tabel 4.1.

Perhitungan untuk bobot *vector* ke-1 dengan data latih pertama:

$$\boldsymbol{D_{1,1}} = \sqrt{\sum_{j=1}^{n}\left(\boldsymbol{X_j} - \boldsymbol{W_{1j}}\right)^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 5{,}943581151)^2 + (3{,}5 - 3{,}03229639)^2 \\ +(1{,}4 - 1{,}844710669)^2 + (0{,}2 - 1{,}090313406)^2]\end{matrix}}$$

$$= 1{,}385929783$$

Perhitungan untuk bobot *vector* ke-2:

$$\boldsymbol{D_{1,2}} = \sqrt{\sum_{j=1}^{n}\left(\boldsymbol{X_j} - \boldsymbol{W_{2j}}\right)^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 5{,}729496669)^2 + (3{,}5 - 2{,}89103111)^2 \\ +(1{,}4 - 1{,}567670545)^2 + (0{,}2 - 1{,}986342912)^2]\end{matrix}}$$

$$= 1{,}996557932$$

Perhitungan untuk bobot *vector* ke-3:

$$D_{1,3} = \sqrt{\sum_{j=1}^{n}(X_j - W_{3j})^2}$$

$$= \sqrt{\begin{matrix}[(5{,}1 - 6{,}393576529)^2 + (3{,}5 - 2{,}969902879)^2 \\ +(1{,}4 - 4{,}014492423)^2 + (0{,}2 - 1{,}849268839)^2]\end{matrix}}$$

$$= 3{,}392639316$$

Dari perhitungan tersebut didapatkan nilai pada Tabel 4.29.

Tabel 4.32 Jarak Bobot dengan Data Latih Pertama

Bobot	jarak
Bobot 1	1,385929783
Bobot 2	1,996557932
Bobot 3	3,392639316

Dari Tabel 4.29 didapatkan bahwa jarak bobot terkecil mendekati kelas 1. Selanjutnya lakukan perhitungan yang sama dengan seluruh data latih pada Tabel 4.1 dengan Persamaan 2.23. dan didapatkan data kedekatan jarak bobot dengan data latih pada Tabel 4.30.

Tabel 4.33 Data Kelas Data Latih dan Bobot

Data latih	Kelas berdasarkan Jarak Bobot Terdekat	Aktual	Keseuaian
1	1	1	1
2	1	1	1
3	3	2	0
4	3	2	0
5	3	3	1
6	3	3	1
Total			4

Dari Tabel 4.30 didapatkan bahwa total data yang sesuai berjumlah 4 dari total 6 data latih. Dengan menggunakan Persamaan 2.24 dan Persamaan 2.25 akan didapatkan nilai akurasi dari proses evaluasi ini.

$$Akurasi = \frac{c}{n}$$
$$= \frac{4}{6}$$
$$= 0{,}666666667$$
$$Akurasi = Akurasi * 100\%$$
$$= 66{,}66667\%$$

4.4 Perancangan Pengujian

Pada tahap perancangan pengujian ini dilakukan untuk menguji bagaimana system yang dibuat akan bekerja dengan baik dan sesuai yang diharapkan ataukah tidak. Proses pengujian sendiri dibagi kedalam beberapa bagian yaitu.

1. Pengujian data dengan *Holdout Method*.
2. Pengujian partikel pada PSO.
3. Pengujian nilai parameter *alpha*.
4. Pengujian jumlah iterasi.

4.4.1 Pengujian Data

Tujuan dari pengujian data dengan *Holdout Method* untuk mengetahui apakah data yang digunakan pada algoritme ini sudah baik atau tidak. Tabel pengujian sendiri berada pada Tabel 4.31.

Tabel 4.34 Pengujian Data dengan *Holdout Method*

No	Perbandingan Data		Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
	Data latih	Data Uji	1	2	3	4	5	
1	140	10						
2	135	15						
3	130	20						
4	125	25						
5	120	30						
6	115	35						
7	110	40						
8	105	45						
9	100	50						
10	95	55						

4.4.2 Pengujian Jumlah Partikel pada PSO

Tujuan dari pegujian jumlah partikel pada PSO ini untuk mengetahui bagaimana pengaruh jumlah partikel pada PSO terhadap kecepatan mendapatkan nilai terbaik pada pengujian PSO yang nantinya akan menjadi nilai awal pada perhitungan LVQ. Pengujian dilakukan dengan menggunakan jumlah particle sebanyak 10, 20, 30, 40, 50, 60, 70, 80, 90 dan 10. Tabel pengujian sendiri pada Tabel 4.32.

Tabel 4.35 Pengujian Jumlah Partikel PSO

No	Jumlah Partikel	Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
		1	2	3	4	5	
1	10						
2	20						
3	30						
4	40						
5	50						
6	60						
7	70						
8	80						
9	90						
10	100						

4.4.3 Pengujian Parameter Alpha pada LVQ

Tujuan dari pengujian parameter alpha pada LVQ ini dilakukan untuk mengetahui bagaimana pengaruh nilai alpha terhadap hasil akurasi dari algoritme LVQ yang digunakan. Nilai *alpha* sendiri yang digunakan adalah 0.1, 0.07, 0.05, 0.03, 0.01, 0.007, 0.005, 0.001, 0.0007 dan 0.0005. Tabel pengujian sendiri dapat dilihat di-Tabel 4.33.

Tabel 4.36 Pengujian Nilai *Alpha*

No	Nilai *Alpha*	Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
		1	2	3	4	5	
1	0,1						
2	0,07						
3	0,05						

4	0,03						
5	0,01						
6	0,007						
7	0,005						
8	0,001						
9	0,0007						
10	0,0005						

4.4.4 Pengujian Jumlah Iterasi

Tujuan dari pengujian iterasi ini dilakukan untuk mengetahui jumlah iterasi terbaik dengan nilai iterasi yang digunakan dari 50, 100, 150, 200, 250, 300, 350, 400, 450 dan 500. Pengujian ini akan dibagi menjadi dua bagian, yaitu pengujian iterasi pada PSO dan juga pengujian iterasi pada LVQ.

4.4.4.1 Pengujian Jumlah Iterasi pada PSO

Tujuan dari pengujian jumlah iterasi pada PSO ini dilakukan untuk mengetahui bagaimana pengaruh iterasi terhadap nilai akurasi pada *global best* (gbest) sebagai bobot awal dari LVQ. Selain itu pengujian ini dilakukan untuk mengetahui apakah terjadi konvergensi dini dari peningkatan iterasi yang ada. Tabel pengujian sendiri dapat dilihat di-Tabel 4.34.

Tabel 4.37 Pengujian Jumlah Iterasi PSO

No		1	2	3	4	5	6	7	8	9	10	11
Iterasi		1	50	100	150	200	250	300	350	400	450	500
Percobaan	1											
	2											
	3											
	4											
	5											

4.4.4.2 Pengujian Jumlah Iterasi pada LVQ

Tujuan dari pengujian jumlah iterasi pada LVQ ini dilakukan untuk mengetahui bagaimana pengaruh iterasi terhadap hasil akurasi dari algoritme LVQ yang digunakan. Tabel pengujian sendiri dapat dilihat pada Tabel 4.35.

Tabel 4.38 Pengujian Jumlah Iterasi LVQ

No	Iterasi	Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
		1	2	3	4	5	
1	50						
2	100						
3	150						
4	200						
5	250						
6	300						
7	350						
8	400						
9	450						
10	500						

BAB 5 IMPLEMENTASI

Bab ini memberikan penjelasan perihal implementasi dari program berdasarkan alur yang telah dibangun pada Bab 4. Implementasi pada bab ini berfokus pada algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) dalam proses penyelesaian klasifikasi data bunga Iris.

5.1 Implementasi Algoritme

Pada sub-bab ini dijelaskan bagaimana pengimplementasian dari metode pada algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) dalam proses klasifikasi data Iris. Berikut akan dijelaskan masing-masing algoritme yang digunakan.

5.1.1 Implementasi Algoritme Pembacaan Data

Algoritme untuk proses pembacaan data ini merupakan proses pembacaan data dari data yang nantinya akan digunakan dalam proses klasifikasi. Data yang digunakan pada mulanya disimpan pada file *excel*, kemudian dilakukan pembacaan data dari *excel* oleh program dan dilakukan penyimpanan data tersebut ke dalam array. Kode program algoritme untuk proses pembacaan data dijelaskan pada Kode Program 5.1.

Kode Program 5.1 Algoritme Pembacaan Data

```
1   public Data() throws IOException, BiffException{
2       File f = new File("E:\\Skripsi\\Dataset.xls");
3       Workbook wb = Workbook.getWorkbook(f);
4       Sheet s = wb.getSheet(0);
5
6       for (int i = 1; i <= total_data; i++) {
7           for (int j = 1; j < 6; j++) {
8               Cell c = s.getCell(j, i);
9               arrData[i-1][j-1] = c.getContents();
10          }
11      }
12  }
```

Pembahasan Kode Program 5.1 sebagai berikut:

1. Baris 2 – 4 : fungsi untuk membaca data pada workbook excel yang akan digunakan dengan menggunakan *library jxl*.
2. Baris 6 – 11 : perulangan untuk membaca data pada workbook.

5.1.2 Implementasi Algoritme *Particle Swarm Optimization*

Dalam algoritme *Particle Swarm Optimization* ini terdapat beberapa proses yang dilakukan seperti proses inisialisasi data, proses *update* kecepatan, *update* partikel, *update* pbest, dan *update* gbest. Proses tersebut dibagi kedalam beberapa sub-program yang akan dijelaskan berikut ini.

5.1.2.1 Implementasi Algoritme Inisialisasi Data Particle Swarm Optimization

Dalam implementasi inisialisasi data dilakukan beberapa proses seperti inisialisasi partikel awal, kecepatan awal, pbest awal dan gbest awal. Berikut algoritme dari inisialisasi data *Particle Swarm Optimization* pada Kode Program 5.2.

Kode Program 5.2 Algoritme Inisialisasi Data *Particle Swarm Optimization*

```
1   private void InisialisasiPSO() {
2       //kecepatan awal
3       int inc = 1;
4       for (double[][] array_kecepatan1 : array_kecepatan) {
5           for (double[] item : array_kecepatan1) {
6               for (int k = 0; k < item.length; k++) {
7                   item[k] = 0;
8               }
9           }
10      }
11
12      /*particle awal
13        hitung akurasi tiap particle
14      */
15
16      inc =1;
17      int a = 0;
18      for (double[][] array_partikel1 : array_partikel) {
19          for (double[] item : array_partikel1) {
20              for (int k = 0; k < item.length; k++) {
21                  item[k] = min_partikel[k] + Math.random() *
22                  (max_partikel[k] - min_partikel[k]);
23              }
24          }
25
26          array_akurasi_partikel[a]=
27          HitungAkurasi(array_partikel1);
28
29          a++;
30      }
31
32      //pbest
33      System.arraycopy(array_partikel, 0, array_pbest, 0,
34      popsize);
35
36      System.arraycopy(array_akurasi_partikel, 0,
37      array_akurasi_pbest, 0, array_akurasi_partikel.length);
38
39      //gbest
40      updategBest();
41  }
```

Pembahasan Kode Program 5.2 sebagai berikut:

1. Baris 3 – 10 : Proses inisialisasi kecepatan awal dengan nilai 0.
2. Baris 16 – 24 : Proses inisialisasi partikel awal secara dengan nilai random.

3. Baris 26 – 27 : Proses menghitung nilai fitness pada setiap partikel dengan fungsi HitungAkurasi.
4. Baris 33 – 34 : Proses inisialisasi pbest awal dengan nilai partikel.
5. Baris 40 : Proses inisialisasi gbest awal dengan fungsi *update*Gbest.

5.1.2.2 Implementasi Algoritme Update Kecepatan

Dalam implementasi algoritme *update* kecepatan dilakukan proses *update* terhadap seluruh kecepatan yang ada berdasarkan nilai bobot inersi, nilai konstanta dan nilai acak dari distribusi *uniform* yang sebelumnya telah diberi nilai. Berikut merupakan implementasi algoritme *update* kecepatan pada Kode Program 5.3.

Kode Program 5.3 Algoritme *Update* Kecepatan

```
1  private void updateKecepatan(int up) {
2      for (int i = 0; i < array_kecepatan.length; i++) {
3          for (int j = 0; j < array_kecepatan[i].length;j++) {
4              for (int k=0;k<array_kecepatan[i][j].length;
5              k++){
6
7                  array_kecepatan[i][j][k] = w *
8                  array_kecepatan[i][j][k] + c1 * r1 *
9                 (array_pbest[i][j][k] –
10                 array_partikel[i][j][k]) + c2 * r2 *
11                (array_gbest[j][k]- array_partikel[i][j][k]);
12
13                 if (array_kecepatan[i][j][k] >
14                 max_kecepatan[k]) {
15                     array_kecepatan[i][j][k] =
16                     max_kecepatan[k];
17                 } else if (array_kecepatan[i][j][k] <
18                 min_kecepatan[k]) {
19                     array_kecepatan[i][j][k] =
20                     min_kecepatan[k];
21                 }
22             }
23         }
24     }
25 }
```

Pembahasan Kode Program 5.3 sebagai berikut:

1. Baris 1 – 25 : Proses *update* kecepatan menggunakan Persamaan 2.19 dengan Batasan nilai kecepatan yang sudah ada.

5.1.2.3 Implementasi Algoritme Update Partikel

Dalam implementasi algoritme *update* partikel dilakukan proses *update* terhadap seluruh partikel yang ada dengan menambahkan nilai partikel sebelumnya dengan nilai kecepatan yang telah di-*update*. Berikut merupakan implementasi algoritme *update* partikel pada Kode Program 5.4.

Kode Program 5.4 Algoritme *Update* Partikel

```
1   private void updateParticle(int up) {
2       for (int i = 0; i < array_partikel.length; i++) {
3           for (int j = 0; j < array_partikel[i].length; j++) {
4               for (int k = 0; k < array_partikel[i][j].length;
5               k++) {
6                   array_partikel[i][j][k] =
7                   array_partikel[i][j][k] +
8                   array_kecepatan[i][j][k];
9
10                  if (array_partikel[i][j][k] >
11                  max_partikel[k]) {
12                      array_partikel[i][j][k] =
13                      max_partikel[k];
14                  } else if (array_partikel[i][j][k] <
15                  min_partikel[k]) {
16                      array_partikel[i][j][k] =
17                      min_partikel[k];
18                  }
19              }
20          }
21          array_akurasi_partikel[i] =
22          HitungAkurasi(array_partikel[i]);
23      }
24  }
```

Pembahasan Kode Program 5.4 sebagai berikut:

Baris 1 – 24 : Proses *update* partikel menggunakan Persamaan 2.20 dengan Batasan nilai kecepatan yang sudah ada.

5.1.2.4 Implementasi Algoritme Update Pbest

Dalam implementasi algoritme *update* pbest dilakukan proses *update* dengan membandingkan nilai *fitness* pada partikel baru dan juga nilai pada pbest sebelumnya, nilai *fitness* pada perbandingan tersebut akan menentukan nilai pbest selanjutnya. Berikut merupakan implementasi algoritme *update* pbest pada Kode Program 5.5.

Kode Program 5.5 Algoritme *Update* Pbest

```
1   private void updatePbest(){
2     for (int i = 0; i < array_akurasi_pbest.length; i++) {
3       if(array_akurasi_partikel[i] >= array_akurasi_pbest[i]){
4            array_pbest[i] = array_partikel[i];
5            array_akurasi_pbest[i] = array_akurasi_partikel[i];
6       }
7     }
8   }
```

Pembahasan Kode Program 5.5 sebagai berikut:

Baris 1 – 8 : Proses *update* pbest dengan membandingkan nilai fitness pbest lama dengan nilai fitness partikel *update*.

5.1.2.5 Implementasi Algoritme Update Gbest

Dalam implementasi algoritme *update* gbest dilakukan proses *update* dengan membandingkan nilai *fitness* pada setiap pbest yang ada, nantinya nilai pbest terbaik akan menjadi nilai gbest selanjutnya. Berikut merupakan implementasi algoritme *update* gbest pada Kode Program 5.6.

Kode Program 5.6 Algoritme *Update* Gbest

```
1  private void updategBest(){
2      double max = Double.MIN_VALUE;
3      int idx = 0;
4      for (int i = 0; i < array_pbest.length; i++) {
5          if (max < array_akurasi_pbest[i]) {
6              idx = i;
7              max = array_akurasi_pbest[i];
8          }
9      }
10     System.arraycopy(array_pbest[idx], 0, array_gbest, 0,
11     array_gbest.length);
12     array_akurasi_gbest = array_akurasi_pbest[idx];
13 }
```

Pembahasan Kode Program 5.6 sebagai berikut:

Baris 1 – 13 : Proses *update* gbest dengan melihan nilai fitness tertinggi pada pbest yang telah di *update*, nantinya pbest tersebut akan menjadi gbest yang baru.

5.1.3 Implementasi Algoritme *Learning Vector Quantization*

Dalam algoritme *Learning Vector Quantization* ini terdapat beberapa proses yang dilakukan seperti proses inisialisasi data dan proses pelatihan data. Proses tersebut dibagi kedalam beberapa sub-program yang akan dijelaskan berikut ini.

5.1.3.1 Implementasi Inisialisasi Data Learning Vector Quantization

Dalam implementasi inisialisasi data pada algoritme *Learning Vector Quantization*, nilai awal bobot berasal dari hasil gbest pada perhitungan algoritme *Particle Swarm Optimization*. Berikut algoritme dari inisialisasi data *Learning Vector Quantization* pada Kode Program 5.7.

Kode Program 5.7 Algoritme Inisialisasi Data

```
1  private void InisialisasiLVQ(){
2      //inisialisasi bobot
3      System.arraycopy(array_gbest, 0, bobot, 0,
4      array_gbest.length);
5  }
```

Pembahasan Kode Program 5.7 sebagai berikut:

Baris 1 – 5 : Nilai bobot awal pada proses lvq adalah hasil gbest terbaik di akhir proses.

5.1.3.2 Implementasi Algoritme Pelatihan Learning Vector Quantization

Dalam implementasi algoritme pelatihan *Learning Vector Quantization* dilakukan dalam beberapa iterasi dan pada setiap iterasi bobot yang ada akan

dilatih dengan seluruh data latih yang telah ditentukan. Sehingga pada hasil akhir akan didapatkan bobot terbaik dari proses pelatihan ini. Untuk algoritme dapat dilihat pada Kode Program 5.8.

Kode Program 5.8 Algoritme Pelatihan

```
for (int i = 0; i < epoch; i++) {
    for (int j = 0; j < Train_data.length; j++) {
        for (int k = 0; k < temp_w.length; k++) {
            temp_w[k] =  Math.sqrt(Math.pow(Train_data[j][0]
            - bobot[k][0], 2) + Math.pow(Train_data[j][1] -
            bobot[k][1], 2) + Math.pow(Train_data[j][2] -
            bobot[k][2], 2) + Math.pow(Train_data[j][3] -
            bobot[k][3], 2));
        }

        winner = Math.min(temp_w[0], Math.min(temp_w[1],
        temp_w[2]));
        runner = winner == temp_w[0] ? Math.min(temp_w[1],
        temp_w[2]) : winner == temp_w[1] ?
        Math.min(temp_w[0], temp_w[2]) : Math.min(temp_w[1],
        temp_w[0]);

        wind = ((winner/runner) > (1-epsilon))&&
        ((runner/winner) < (1+epsilon));

        runcek = (runner == temp_w[0] &&
        Train_data[j][4] == 1) || (runner == temp_w[1] &&
        Train_data[j][4] == 2) ||(runner == temp_w[2] &&
        Train_data[j][4] == 3);

        if (wind && runcek){
            if (winner == temp_w[0]) {
                if (runner == temp_w[1]) {
                    for (int k = 0; k < 4; k++) {
                        bobot[0][k] = bobot[0][k] - alpha *
                        (Train_data[j][k] - bobot[0][k]);
                        bobot[1][k] = bobot[1][k] + alpha *
                        (Train_data[j][k] - bobot[1][k]);
                    }
                }else if(runner == temp_w[2]){
                    for (int k = 0; k < 4; k++) {
                        bobot[0][k] = bobot[0][k] - alpha *
                        (Train_data[j][k] - bobot[0][k]);
                        bobot[2][k] = bobot[2][k] + alpha *
                        (Train_data[j][k] - bobot[2][k]);
                    }
                }
            }else if (winner == temp_w[1]) {
                if (runner == temp_w[0]) {
                    for (int k = 0; k < 4; k++) {
                        bobot[1][k] = bobot[1][k] - alpha *
                        (Train_data[j][k] - bobot[1][k]);
                        bobot[0][k] = bobot[0][k] + alpha *
                        (Train_data[j][k] - bobot[0][k]);
                    }
                }else if(runner == temp_w[2]){
                    for (int k = 0; k < 4; k++) {
```

```
                        bobot[1][k] = bobot[1][k] - alpha *
                        (Train_data[j][k] - bobot[1][k]);
                        bobot[2][k] = bobot[2][k] + alpha *
                        (Train_data[j][k] - bobot[2][k]);
                    }
                }
            }else if (winner == temp_w[2]) {
                if (runner == temp_w[0]) {
                    for (int k = 0; k < 4; k++) {
                        bobot[2][k] = bobot[2][k] - alpha *
                        (Train_data[j][k] - bobot[2][k]);
                        bobot[0][k] = bobot[0][k] + alpha *
                        (Train_data[j][k] - bobot[0][k]);
                    }
                }else if(runner == temp_w[1]){
                    for (int k = 0; k < 4; k++) {
                        bobot[2][k] = bobot[2][k] - alpha *
                        (Train_data[j][k] - bobot[2][k]);
                        bobot[1][k] = bobot[1][k] + alpha *
                        (Train_data[j][k] - bobot[1][k]);
                    }
                }
            }
        }else{
            if (winner == temp_w[0]) {
                if (Train_data[j][4] == 1) {
                    for (int k = 0; k < 4; k++) {
                        bobot[0][k] = bobot[0][k] + alpha *
                        (Train_data[j][k] - bobot[0][k]);
                    }
                }else{
                    for (int k = 0; k < 4; k++) {
                        bobot[0][k] = bobot[0][k] - alpha *
                        (Train_data[j][k] - bobot[0][k]);
                    }
                }
            }else if (winner == temp_w[1]) {
                if (Train_data[j][4] == 2) {
                    for (int k = 0; k < 4; k++) {
                        bobot[1][k] = bobot[1][k] + alpha *
                        (Train_data[j][k] - bobot[1][k]);
                    }
                }else{
                    for (int k = 0; k < 4; k++) {
                        bobot[1][k] = bobot[1][k] - alpha *
                        (Train_data[j][k] - bobot[1][k]);
                    }
                }
            }else if (winner == temp_w[2]) {
                if (Train_data[j][4] == 3) {
                    for (int k = 0; k < 4; k++) {
                        bobot[2][k] = bobot[2][k] + alpha *
                        (Train_data[j][k] - bobot[2][k]);
                    }
                }else{
                    for (int k = 0; k < 4; k++) {
                        bobot[2][k] = bobot[2][k] - alpha *
                        (Train_data[j][k] - bobot[2][k]);
                    }
```

```
112                     }
113                 }
114             }
115         }
116         alpha = init * (1 - (double) i / (double) epoch);
117     }
```

Pembahasan Kode Program 5.8 sebagai berikut:

Baris 1 – 117 : Perulangan sebanyak iterasi/epoch sebagai proses pelatihan.

Baris 2 – 115 : Perulangan sebanyak jumlah data uji.

Baris 3 – 9 : Proses perhitungan jarak bobot dengan data uji ke-*j*.

Baris 11 – 24 : Pencarian jarak terbaik pertama dan kedua yang selanjutnya dilakukan pengecekan apakah sesuai dengan persyaratan untuk *update*.

Baris 26 – 76 : Apabila hasil pengecekan jarak bobot pertama dan kedua sesuai dengan persyaratan maka dilakukan *update* terhadap bobot pada bobot dengan jarak terpendek pertama dan kedua.

Baris 76 – 115 : Apabila hasil pengecekan bobot tidak sesuai maka dilakukan *update* terhadap bobot terpendek pertama.

Baris 116 : *Update* nilai alpha dengan nilai baru.

5.1.4 Implementasi Algoritme untuk Pengujian Hasil

Dalam algoritme pengujian ini digunakan untuk mengetahui nilai akurasi dari proses perhitungan yang telah dilakukan oleh algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ). Selain digunakan untuk mengetahui hasil akhir dari perhitungan dari algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ), algoritme ini juga digunakan untuk mengetahui nilai fitness pada partikel pada algoritme *Particle Swarm Optimization.* Untuk algoritme itu sendiri dapat dilihat pada Kode Program 5.9.

Kode Program 5.9 Algoritme Pengujian Hasil

```
 1  double HitungAkurasi(double arr[][]) {
 2      int benar = 0;
 3      int total = jumlah_data_uji;
 4      double c;
 5      double temp[] = new double[3];
 6
 7      for (double[] Test : Testing_data) {
 8          for (int j = 0; j < arr.length; j++) {
 9              temp[j] = Math.sqrt(Math.pow(Test[0] -
10              arr[j][0], 2) + Math.pow(Test[1]-arr[j][1], 2) +
11              Math.pow(Test[2] - arr[j][2], 2) +
12              Math.pow(Test[3] - arr[j][3], 2));
13          }
14          c = Math.min(temp[0], Math.min(temp[1], temp[2]));
```

```
15          if (c == temp[0] && Test[4] == 1) {
16              benar++;
17          } else if (c == temp[1] && Test[4] == 2) {
18              benar++;
19          } else if (c == temp[2] && Test[4] == 3) {
20              benar++;
21          }
22      }
23      return (double) benar / total;
24  }
```

Pembahasan Kode Program 5.9 sebagai berikut:

Baris 2 – 5 : Parameter nilai untuk proses pengujian.

Baris 6 – 22 : Proses pengujian dengan membandingkan bobot dan data uji.

Baris 23 : Memberikan hasil pengujian kepada parameter yang memanggil fungsi ini.

BAB 6 ANALISIS DAN PENGUJIAN

Bab 6 ini menjelaskan tentang analisis dan pengujian terhadap algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) dalam proses klasifikasi data Iris. Pengujian yang dilakukan meliputi pengujian data, pengujian partikel pada *Particle Swarm Optimization* (PSO), pengujian parameter alpha dan jumlah iterasi pada *Learning Vector Quantization* (LVQ).

6.1 Pengujian Data

Pengujian data ini menjelaskan bagaimana data yang digunakan pada algoritme ini sudah baik atau tidak.

6.1.1 Skenario Pengujian Data

Pengujian data ini menggunakan metode *Holdout Method* dengan membagi data menjadi dua, yaitu data latih dan data uji. Pengujian ini akan dilakukan sebanyak 5 kali dan juga akan dilihat bagaimana rata-rata secara keseluruhan algoritme ini. Selain itu pengujian ini menggunakan nilai parameter berikut:

- *Particle Swarm Optimization*
  - Jumlah partikel : 10
  - Iterasi : 50
  - w : 0,5
  - $c1$ dan $c2$: 0,7
  - $r1$ dan $r2$: 0,4
- *Learning Vector Quantization*
  - Iterasi : 50
  - *Alpha* (α) : 0,01
  - *Epsilon* (ε) : 0,35

Hasil pengujian data terhadap algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) berada pada Tabel 6.1.

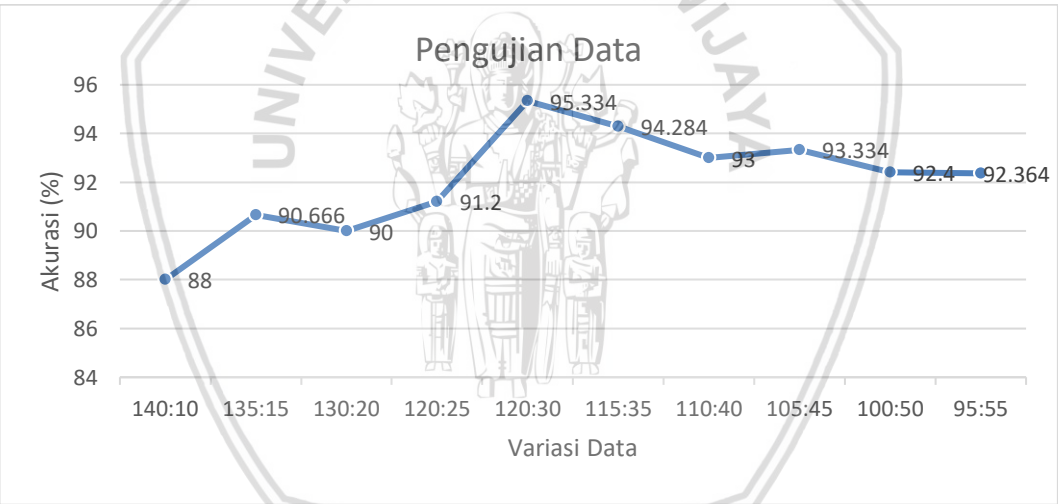
Tabel 6.1 Hasil Pengujian Data

No	Perbandingan Data		Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
	Data latih	Data Uji	1	2	3	4	5	
1	140	10	70	80	100	100	90	88
2	135	15	86.67	93.33	93.33	100	80	90.666
3	130	20	90	90	95	85	90	90

4	125	25	100	92	92	92	80	91.2
5	120	30	83.33	100	96.67	100	96.67	95.334
6	115	35	88.57	100	88.57	97.14	97.14	94.284
7	110	40	95	90	95	92.5	92.5	93
8	105	45	97.78	88.89	95.56	93.33	91.11	93.334
9	100	50	94	94	94	88	92	92.4
10	95	55	85.45	94.54	96.37	89.09	96.37	92.364

6.1.2 Analisis Pengujian Data

Berdasarkan Tabel 6.1 diketahui perbandingan data latih dan data uji yang memiliki rata-rata akurasi tertinggi berada pada ke- 5 dengan rata-rata akurasinya adalah 95.334 persen dengan akurasi tertinggi 100 persen dan akurasi terendah 83.33 persen. Sehingga perbandingan data yang akan digunakan pada pengujian berikutnya adalah ke- 5 dengan perbandingan data 120:30. Untuk hasil pengujian dapat dilihat pada Gambar 6.1.



Gambar 6.1 Grafik Akurasi Hasil Pengujian Data

Dari Gambar 6.1 didapatkan bahwa nilai rata-rata tertinggi ada pada pengujian ke-5 dengan rata-rata 95,334 persen dan rata-rata terendah pada pengujian ke-1 dengan rata-rata 88. Pengujian ke-1 memiliki nilai terendah dikarenakan data uji yang digunakan sedikit, hal tersebut yang membuat nilai rata-rata akurasinya rendah begitu juga apabila data latihnya yang sedikit, akan menyebabkan rendahnya nilai akurasi dikarenakan kurangnya proses pelatihan terhadap bobot data. Hal tersebut dapat dilihat pada Gambar 6.1 bahwa semakin berkurangnya data uji menyebabkan turunnya akurasi.

6.2 Pengujian Jumlah Partikel pada PSO

Pengujian jumlah partikel ini menjelaskan bagaimana pengaruh partikel pada *Particle Swarm Optimization* (PSO) terhadap nilai akhir dari akurasi pada algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ).

6.2.1 Skenario Pengujian Jumlah Partikel

Pengujian partikel ini dilakukan untuk melihat skenario partikel mana yang menghasilkan hasil akurasi terbaik. Pada pengujian ini akan digunakan nilai parameter berikut:

- *Particle Swarm Optimization*
  - Iterasi : 50
  - w : 0,5
  - $c1$ dan $c2$: 0,7
  - $r1$ dan $r2$: 0,4
- *Learning Vector Quantization*
  - Iterasi : 50
  - *Alpha* (α) : 0,01
  - *Epsilon* (ε) : 0,35

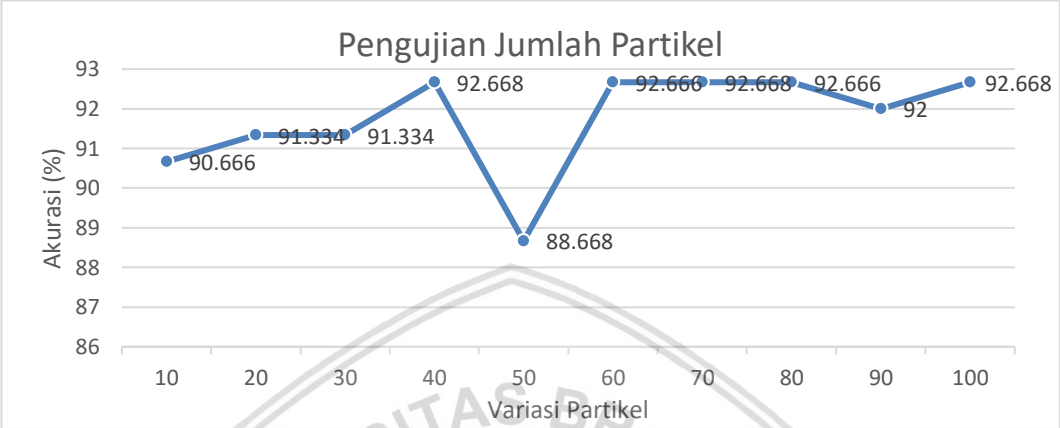
Hasil pengujian partikel PSO terhadap algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) berada pada Tabel 6.2.

Tabel 6.2 Hasil Pengujian Jumlah Partikel PSO

No	Jumlah Partikel	Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
		1	2	3	4	5	
1	10	90	93,33	83,33	90	96,67	90,666
2	20	96,67	90	86,67	90	93,33	91,334
3	30	96,67	90	93,33	90	86,67	91,334
4	40	86,67	86,67	100	93,33	96,67	92,668
5	50	70	96,67	86,67	93,33	96,67	88,668
6	60	83,33	100	96,67	90	93,33	92,666
7	70	96,67	100	86,67	90	90	92,668
8	80	90	93,33	86,67	100	93,33	92,666
9	90	93,33	100	86,67	90	90	92
10	100	86,67	96,67	90	93,33	96,67	92,668

6.2.2 Analisis Pengujian Jumlah Partikel

Berdasarkan Tabel 6.2 didapatkan bahwa rata-rata akurasi tertinggi yaitu 92,668 persen dengan nilai partikel 40 dan 100, sehingga pada pengujian selanjutnya nilai partikel yang digunakan adalah 40. Untuk hasil percobaan jumlah partikel dapat dilihat pada Gambar 6.2.



Gambar 6.2 Grafik Hasil Pengujian Jumlah Partikel

Dari pengujian jumlah partikel yang telah dilakukan, didapatkan nilai akurasi terendah yang bernilai 70 persen pada percobaan variasi partikel 50, hal tersebut terjadi dikarenakan nilai random yang dihasilkan dari setiap partikel memiliki nilai yang kurang bagus sehingga menghasilkan akurasi yang kurang bagus juga. Jumlah partikel sendiri berpengaruh terhadap variasi nilai yang dihasilkan dalam proses PSO, semakin banyak partikel yang digunakan semakin besar kemungkinan hasil yang dihasilkan semakin baik, namun dengan jumlah partikel yang banyak dapat menyebapkan proses perhitungan melambat dikarenakan banyaknya partikel yang harus diproses.

6.3 Pengujian Parameter *Alpha* pada LVQ

6.3.1 Skenario Pengujian Parameter *Alpha*

Pengujian nilai *alpha* ini dilakukan untuk melihat skenario nilai *alpha* mana yang menghasilkan hasil akurasi terbaik. Pada pengujian ini akan digunakan nilai parameter berikut:

- *Particle Swarm Optimization*
  - Iterasi : 50
  - Partikel : 40
  - w : 0,5
  - $c1$ dan $c2$: 0,7
  - $r1$ dan $r2$: 0,4

- *Learning Vector Quantization*
  - Iterasi : 50
  - *Epsilon* (ε) : 0,35

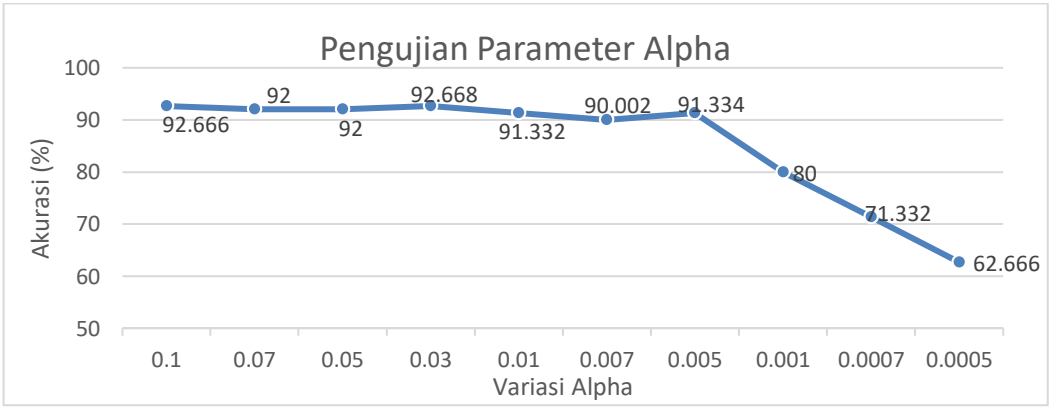
Hasil pengujian nilai alpha terhadap algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) pada Tabel 6.3.

Tabel 6.3 Hasil Pengujian Nilai *Alpha*

No	Nilai *Alpha*	Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
		1	2	3	4	5	
1	0,1	100	86,67	93,33	93,33	90	92,666
2	0,07	96,67	93,33	90	93,33	86,67	92
3	0,05	100	90	93,33	86,67	90	92
4	0,03	90	96,67	86,67	90	100	92,668
5	0,01	96,67	93,33	93,33	90	83,33	91,332
6	0,007	96,67	80	90	86,67	96,67	90,002
7	0,005	80	96,67	90	96,67	93,33	91,334
8	0,001	70	96,67	90	50	93,33	80
9	0,0007	50	73,33	60	80	93,33	71,332
10	0,0005	30	66,67	83,33	63,33	70	62,666

6.3.2 Analisis Pengujian Parameter *Alpha*

Berdasarkan Tabel 6.3 diketahui bahwa rata-rata akurasi tertinggi sebesar 92,668 persen dengan nilai akurasi tertinggi sebesar 100 persen pada pengujian dengan nilai *alpha* 0,03. Dengan hasil tersebut maka nilai *alpha* yang akan digunakan pada pengujian selanjutnya adalah 0,03. Hasil akurasi pengujian parameter alpha dijelaskan pada Gambar 6.3.



Gambar 6.3 Grafik Hasil Pengujian Parameter *Alpha*

Hasil pengujian tersebut menunjukkan bahwa pengaruh nilai *alpha* cukup signifikan dalam mempengaruhi hasil akurasi, dimana nilai *alpha* yang semakin kecil dapat mengakibatkan nilai akurasi pada LVQ menjadi menurun. Hal tersebut ditunjukkan penurunan akurasi pada percobaan dengan nilai *alpha* 0,005 dimana akurasi sebesar 91,334% mengalami penurunan menjadi 80% dengan nilai *alpha* 0,001 dan terus menurun seiring mengecilnya nilai *alpha*.

6.4 Pengujian Jumlah Iterasi LVQ

6.4.1 Skenario Pengujian Jumlah Iterasi PSO

Pengujian jumlah iterasi ini dilakukan untuk melihat skenario mana yang menghasilkan hasil akurasi terbaik. Pada pengujian ini akan digunakan nilai parameter berikut:

- *Particle Swarm Optimization*
  - Partikel : 40
  - w : 0,5
  - $c1$ dan $c2$: 07
  - $r1$ dan $r2$: 0,4
- *Learning Vector Quantization*
  - Iterasi : 50
  - *Alpha* (α) : 0,03
  - *Epsilon* (ε) : 0,35

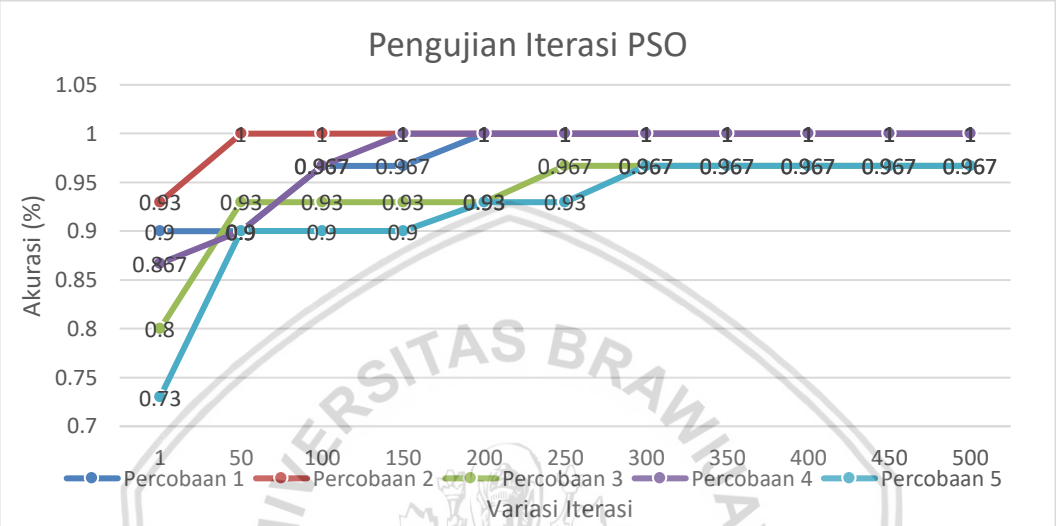
Hasil pengujian iterasi pada PSO terhadap algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) pada Tabel 6.4.

Tabel 6.4 Hasil Pengujian Jumlah Iterasi PSO

No		1	2	3	4	5	6	7	8	9	10	11
Iterasi		1	50	100	150	200	250	300	350	400	450	500
Percobaan	1	0,9	0,9	0,967	0,967	1	1	1	1	1	1	1
	2	0,933	1	1	1	1	1	1	1	1	1	1
	3	0,8	0,93	0,933	0,933	0,933	0,967	0,967	0,967	0,967	0,967	0,967
	4	0,867	0,9	0,967	1	1	1	1	1	1	1	1
	5	0,73	0,9	0,9	0,9	0,933	0,933	0,967	0,967	0,967	0,967	0,967

6.4.2 Analisis Pengujian Jumlah Iterasi PSO

Dari pengujian pada Tabel 6.4 didapatkan bahwa iterasi ke-300 hingga iterasi ke-500 tidak terjadi perubahan nilai gbest. Sehingga pada pengujian selanjutnya digunakan iterasi pada PSO adalah 300. Untuk hasil pengujian dapat dilihat pada Gambar 6.4.



Gambar 6.4 Grafik Hasil Pengujian Iterasi PSO

Dari Gambar 6.4 dapat terlihat, perubahan nilai akurasi pada gbest mulai menghasilkan nilai yang stabil pada iterasi ke 300 pada setiap percobaan. Dari percobaan yang telah dilakukan dapat dilihat bahwa nilai yang dihasilkan oleh gbest yang nantinya akan dijadikan sebagai bobot awal memiliki nilai akurasi 1 dan juga 0,967.

6.4.3 Skenario Pengujian Jumlah Iterasi LVQ

Pengujian jumlah iterasi ini dilakukan untuk melihat skenario mana yang menghasilkan hasil akurasi terbaik. Pada pengujian ini akan digunakan nilai parameter berikut:

- *Particle Swarm Optimization*
  - Iterasi : 300
  - Partikel : 40
  - w : 0,5
  - $c1$ dan $c2$: 07
  - $r1$ dan $r2$: 0,4
- *Learning Vector Quantization*
  - *Alpha* (α) : 0,03
  - *Epsilon* (ε) : 0,35

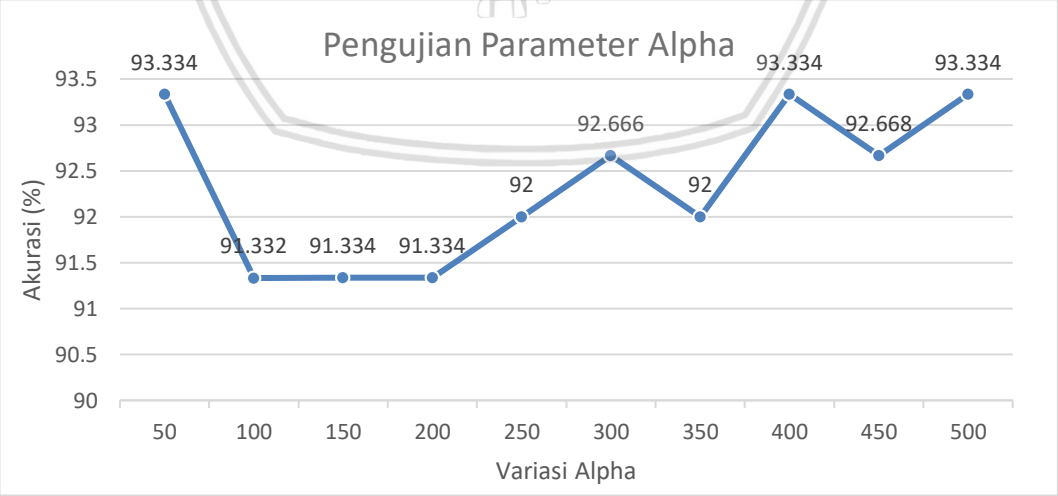
Hasil pengujian iterasi pada LVQ terhadap algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) berada pada Tabel 6.5.

Tabel 6.5 Hasil Pengujian Jumlah Iterasi LVQ

No	Iterasi	Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
		1	2	3	4	5	
1	50	100	93,33	96,67	90	86,67	93,334
2	100	83,33	93,33	96,67	93,33	90	91,332
3	150	90	90	96,67	86,67	93,33	91,334
4	200	86,67	96,67	93,33	86,67	93,33	91,334
5	250	86,67	93,33	96,67	93,33	90	92
6	300	93,33	93,33	96,67	90	90	92,666
7	350	93,33	96,67	90	93,33	86,67	92
8	400	90	90	96,67	100	96,67	93,334
9	450	86,67	96,67	96,67	93,33	90	92,668
10	500	96,67	100	90	86,67	93,33	93,334

6.4.4 Analisis Pengujian Jumlah Iterasi LVQ

Dari pengujian pada Tabel 6.5 didapatkan bahwa nilai rata-rata akurasi tertinggi adalah 93,334 persen dengan nilai iterasi (*epoch*) adalah 50 dan 500. Namun nilai iterasi terbaiknya adalah 50 dikarenakan bernilai lebih kecil dari iterasi yang lain. Hasil pengujian iterasi dapat dilihat pada Gambar 6.5.



Gambar 6.5 Grafik Hasil Pengujian Iterasi LVQ

Hasil pengujian didapatkan bahwa perubahan iterasi tidak begitu berpengaruh pada hasil percobaan iterasi ini. Hal tersebut terjadi dikarenakan nilai bobot yang merupakan hasil perhitungan gbest dapat membantu meningkatkan akurasi dari proses ini. Namun jumlah iterasi berpengaruh terhadap lamanya proses pelatihan, karena semakin banyak proses iterasi yang dilakukan membutuhkan waktu untuk memprosesnya.

6.5 Analisis Global

Pada analisis global ini akan dilakukan perbandingan dari algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) dengan algoritme *Learning Vector Quantization* (LVQ). Hal ini dilakukan agar dapat diketahui hasil dari algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) apakah lebih baik ataukah tidak terhadap algoritme *Learning Vector Quantization* (LVQ). Dalam proses perbandingan ini akan dilakukan dengan tiga pengujian yaitu pengujian nilai *alpha*, jumlah iterasi pada algoritme *Learning Vector Quantization* (LVQ) dan juga time excecution. Untuk jumlah data latih yang digukanan sebanyak 75 data latih, begitu juga untuk data uji yang digunakan sebanyak 75. Selain itu nilai parameter yang digunakan sama dengan menggunakan nilai parameter terbaik dari pengujian sebelumnya yaitu:

- *Particle Swarm Optimization*
  - Iterasi : 300
  - Partikel : 40
  - w : 0,5
  - $c1$ dan $c2$: 0,7
  - $r1$ dan $r2$: 0,4
- *Learning Vector Quantization*
  - Iterasi : 50
  - *Alpha* (α) : 0,03
  - *Epsilon* (ε) : 0,35

Dengan menggunakan parameter diatas, didapatkan hasil pada Tabel 6.6 dan Tabel 6.6 berikut.

Tabel 6.6 Hasil Perbandingan Pengujian Nilai *Alpha* dan Iterasi

Algoritme	Percobaan ke-*i* (%)					Rata-Rata Akurasi (%)
	1	2	3	4	5	
PSO-LVQ	89,33	94,67	94,67	90,67	97,33	93,334
LVQ	96	90,67	88	60	86,67	84,268
Selisih						9,066

Tabel 6.7 Hasil *Time Excecution* dari Kedua Algoritme

Algoritme	Percobaan ke-*i* (milisecond)					Rata-Rata
	1	2	3	4	5	
PSO-LVQ	286	285	306	322	301	300
LVQ	158	153	151	156	153	154,2

Dari hasil diatas didapatkan bahwa nilai akurasi pada algoritme LVQ mengalami peningkatan dimana akurasi terbaik pada algoritme PSO-LVQ adalah 97,33% dan pada algoritme LVQ sendiri adalah 96%. Pada algoritme LVQ sendiri terdapat nilai akurasi yang rendah yaitu 60 %, hal tersebut disebabkan nilai bobot awal yang diperoleh merupakan nilai bobot awal yang kurang bagus sehingga nilai akurasi yang diperoleh juga tidak sesuai harapan. Namun walaupun dapat mengoptimalkan nilai akurasi dari algortime LVQ, algoritme PSO-LVQ memiliki waktu eksekusi yang cukup lama dibandingkan dengan algoritme LVQ pada umumnya yaitu 300 *milisecond* atau sekitar 0,3 detik, berbeda dengan LVQ yang memiliki waktu eksekusi 154,2 *milisecond* atau sekitar 0,15 detik.

BAB 7 PENUTUP

Pada bab ini menjelaskan kesimpulan dan juga saran dari penelitian yang telah dilakukan berdasarkan perancangan, implementasi serta analisis dan pengujian yang telah dilakukan.

7.1 Kesimpulan

Berdasarkan penelitian yang berjudul "Penerapan Algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) pada Klasifikasi Data Iris", maka dapat disimpulkan bahwa:

1. Algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) untuk klasifikasi data *Iris* dapat diimplementasikan dengan baik. Langkah dalam proses penerapan algoritme sendiri diawali dengan melakukan pembagian data yang sebelumnya diambil dari website UCI *Machine Learning,* data tersebut dibagi menjadi dua bagian yaitu data latih dan data uji dengan menggunakan metode *Holdout Method*. Selanjutnya dilakukan proses penentuan bobot awal sebagai kelas prediksi dengan algoritme PSO untuk mendapatkan bobot terbaik yang nantinya digunakan pada algoritme LVQ. Setelah mendapatkan bobot terbaik, dilakukan proses pelatihan pada algoritme LVQ sehingga didapatkan bobot prediksi dari setiap kelas. Hasil bobot tersebut kemudian diuji dengan menggunakan data uji yang telah dibuat sebelumnya berupa nilai akurasi yang diperoleh dengan melakukan perhitungan kedekatan data uji dengan bobot hasil perhitungan.
2. Hasil akurasi yang diperoleh dari algoritme *Particle Swarm Optimization – Learning Vector Quantization* (PSO-LVQ) berdasarkan perbandingan data uji dan data latih sebesar 120:30 didapatkan rata-rata akurasi 95,334%. Selain itu nilai akurasi berdasarkan jumlah partikel = 40, w = 0,5, c1 = 0,7, c2 = 0,7, r1 = 0,4, r2 = 0,4, iterasi PSO = 300, nilai *alpha* = 0,05, iterasi LVQ = 50, dan epsilon = 0,35 diperoleh rata-rata 93,334% dengan akurasi tertinggi 100%. Selain itu hasil perbandingan dengan algoritme LVQ didapatkan bahwa algoritme PSO-LVQ mampu mengoptimalkan akurasi dari LVQ dimana akurasi rata-rata PSO-LVQ sebesar 93,334% dan LVQ sebesar 84,268% dengan selisih 9,066%. Namun dari segi waktu ekseskusi waktu yang dibutuhkan PSO-LVQ lebih lama dengan rata-rata waktu eksekusi 0,3 detik berbeda dengan LVQ yang membutuhkan waktu eksekusi 0,15 detik.

7.2 Saran

Proses klasifikasi ini masih memiliki kekurangan, saran yang mampu diberikan untuk pengembangan klasifikasi dengan algoritme ini kedepannya yaitu:

1. Pada penelitian selanjutnya diharapkan dapat melakukan optimasi bobot dengan algoritme lain sehingga akan didapat hasil akurasi yang

lebih baik. Karena hasil akurasi yang didapatkan dari gbest pada proses PSO dipengaruhi jumlah partikel dan juga jumlah iterasi pada PSO sehingga perlu dilakukan pencarian terlebih dahulu pada kedua nilai tersebut.

2. Pada penelitian selanjutnya dapat dilakukan pengujian terhadap nilai *epsilon* sehingga dapat diketahui pengaruh epsilon terhadap tingkat akurasi dari LVQ yang digunakan.
3. Pada penelitian selanjutnya dapat dikembangkan dengan variasi LVQ yang lain seperti LVQ3. Karena pada LVQ3 proses pelatihan juga dipengaruhi oleh nilai epsilon sehingga akan didapatkan hasil yang lebih baik.

DAFTAR PUSTAKA

Arniantya, R., Setiawan, B. D. & Adikara, P. P., 2018. Optimasi Vektor Bobot Pada *Learning Vector Quantization* Menggunakan Algoritme Genetika Untuk Identifikasi Jenis *Attention Deficit Hyperactivity Disorder* Pada Anak. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* Volume 2, pp. 679-687.

Azmi, M., 2014. Komparasi Metode Jaringan Syaraf Tiruan SOM dan LVQ Untuk Mengidentifikasi Data Bunga Iris. *Jurnal TEKNOIF,* Volume 2, pp. 64-70.

Fausett, L. V., 1994. *Fudamentals of Neural Networks (Architectures, Algorithms, and Applications.* New Jersey: Prentice-Hall.

Fisher, R. A., 2018. *Iris Data Set.* [Online] Available at: https://archive.ics.uci.edu/ml/datasets/iris [Accessed 20 January 2018].

Haldar, R. & Mishra, P. K., 2016. *Learning Vector Quantization (LVQ) Neural Network Approach for Multilingual Speech Recognition*. *International Research Journal of Engineering and Technology (IRJET),* 03(05), pp. 2863-2869.

Han, J. & Kamber, M., 2006. *Data Mining: Concepts and Techniques.* 2nd ed. San Francisco: Elsevier Inc.

Hardinata, J. T. et al., 2017. *Modification Of Learning Rate With Lvq Model Improvement In Learning Backpropagation.* Medan, IOPSCIENCE.

Haykin, S., 2001. *Neural Networks: A Comprehensive Foundation.* 2nd ed. Ontario: Pearson Education Inc.

Jatmiko, W. et al., 2009. *Fuzzy Learning Vector Quantization Based on Particle Swarm Optimization For Artificial Odor Dicrimination System*. *WSEAS TRANSACTIONS on SYSTEMS,* 8(12), pp. 1239-1252.

Kamenestky, R. & Hiroshi, O., 2013. *Ornamental Geophytes: From Basic Science to Sustainable Production.* Boca Raton: CRC Press.

Kennedy, J. & Eberhart, E., 1995. *Particle swarm optimization.* Perth, WA, Australia, IEEE Xplore.

Rahardian, B. A., Dewi, C. & Rahayudi, B., 2018. Implementasi *Genetic Algorithm* Dan *Artificial Neural Network* Untuk Deteksi Dini Jenis *Attention Deficit Hyperactivity Disorder*. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer ,* Volume 2, pp. 688-694.

Swain, M., Dash, S. K., Dash, S. & Mohapatra, A., 2012. *An Approach For Iris Plant Classification Using Neural Network*. *International Journal on Soft Computing,* Volume 3, pp. 79-89.

Tan, H. T. W. & Xingli, G., 2008. *Plant Magic: Auspicious and Inauspicious Plants from Around the World.* 1st ed. Singapura: Marshall Cavendish International (Asia).

Wuryandari, M. D. & Afrianto, I., 2012. Perbandingan Metode Jaringan Syaraf Tiruan *Backpropagation* dan *Learning Vector Quantization* pada Pengenalan Wajah. *Jurnal Komputer dan Informatika (KOMPUTA),* Volume 1, pp. 45-51.